

Architektur des Oracle Application-Server 4.0.8 und seine Einsatzmöglichkeiten für HTTP-, EJB- und CORBA-Anwendungen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Wirtschaftsinformatiker

an der

Fachhochschule Köln
Abteilung Gummersbach
Fachbereich Informatik
Studiengang Wirtschaftsinformatik

1. Prüfer: Prof. Dr. rer. nat. Heide Faeskorn-Woyke
2. Prüfer: Prof. Dr. rer. nat. Dipl.-Phys. Peter Göttel

Erstellt von:

Michael Zeidler
Altendahl 9
51515 Kürten
Matrikel-Nr.: 1046192111

Oliver Hüskes
Dellbrücker Straße 104
51469 Bergisch Gladbach
Matrikel-Nr.: 1101470118

Gummersbach, 30. Juli 2000

Inhaltsverzeichnis

	Seite
Anhangverzeichnis	III
Abbildungsverzeichnis	IV
Tabellenverzeichnis	VIII
1 Einleitung	1
2 Grundlagen	3
2.1 TCP/IP	4
2.2 World Wide Web	6
2.3 Netzwerkarchitekturen	9
2.4 CORBA	14
2.5 Enterprise JavaBeans	22
2.5.1 Überblick	22
2.5.2 Architektur	23
3 Oracle Application-Server	30
3.1 Einführung	30
3.2 Architektur	33
3.2.1 HTTP-Listener-Ebene	36
3.2.2 OAS-Ebene	38
3.2.3 Anwendungsebene	40
3.2.3.1 Cartridge	40
3.2.3.2 Anwendung	44
3.2.3.2.1 Cartridge-basierte Anwendung	44
3.2.3.2.2 Komponenten-basierte Anwendung	46
3.2.4 Skalierbarkeit und Lastverteilung	49
3.3 OAS-Manager	50
3.4 Datenbankzugriff	53
3.5 Interaktionsmodelle	63

3.5.1	Anfrage/Antwort-Interaktion	63
3.5.2	Sitzung.....	64
3.5.3	Transaktion.....	66
3.5.3.1	Deklarative Transaktion	70
3.5.3.2	Programmtechnische Transaktion	75
3.6	Sicherheit	78
3.6.1	Authentifizierung	78
3.6.2	Datenverschlüsselung.....	88
3.6.3	Sicherheit bei PL/SQL-Anwendungen.....	92
3.7	Einsatzmöglichkeiten	93
3.7.1	PL/SQL-Anwendungen.....	93
3.7.2	Perl- und LiveHTML-Anwendungen.....	98
3.7.3	JServlet-Anwendungen	102
3.7.4	C++-CORBA-Anwendungen.....	108
3.7.5	EJB-Anwendungen	118
3.7.6	Vergleich der Einsatzmöglichkeiten	130
4	Beschreibung der Anwendung "OnlineTicket"	132
5	Schlußbetrachtung.....	143
Anhang		146
Literaturverzeichnis		192

Anhangverzeichnis

	Seite
Anlage 1: Glossar	147
Anlage 2: Quellcode der Klasse "AppletTicketShop"	150
Anlage 3: Quellcode der Klasse "PanelVeranstaltungAuswaehlen"	153
Anlage 4: Quellcode der Klasse "PanelPlatzwahl"	157
Anlage 5: Quellcode der Klasse " PanelBuchungsdatenErfassen "	162
Anlage 6: Quellcode der Klasse "genTableModel"	169
Anlage 7: Quellcode der Klasse "MessageDialog"	171
Anlage 8: Quellcode des Home-Interface "TicketHome"	173
Anlage 9: Quellcode des Remote-Interface "TicketRemote"	173
Anlage 10: Quellcode der Bean-Klasse "Ticket"	175
Anlage 11: HTML-Dokument zum Laden des Applet	188
Anlage 12: Quellcode der PL/SQL-Prozedur "delete_Reservierungen"	188
Anlage 13: Skript zum Erzeugen des Datenbank-Schema	188

Abbildungsverzeichnis

	Seite
Abbildung 1: Schichten der Rechner-Kommunikation über TCP/IP.....	4
Abbildung 2: Aufbau einer URL.....	7
Abbildung 3: Beispiel einer URL	7
Abbildung 4: Kommunikation über HTTP zwischen Client und Server	8
Abbildung 5: Anordnungen der Komponenten einer Anwendung	10
Abbildung 6: Mainframe-Architektur	11
Abbildung 7: 2-Schicht-Client/Server-Architektur.....	12
Abbildung 8: 3-Schicht-Client/Server-Architektur.....	13
Abbildung 9: OMA-Referenzmodell	15
Abbildung 10: Schema der Proxy-Kommunikation.....	17
Abbildung 11: Anfrage über den ORB	19
Abbildung 12: Struktur eines ORB	20
Abbildung 13: EJB-Architektur	23
Abbildung 14: Stateless und stateful Session-Bean.....	26
Abbildung 15: Aufgaben eines Application-Server.....	32
Abbildung 16: Komponenten des Primary-Node einer OAS-Web-Site	34
Abbildung 17: Bearbeitung einer Anfrage an eine Anwendung durch den OAS	34
Abbildung 18: OAS-Architektur.....	35
Abbildung 19: Cartridge-Server und Cartridge-Instanzen.....	42
Abbildung 20: Thread-Modelle der Cartridge-Server	43
Abbildung 21: Anfrage an eine cartridge-basierte Anwendung	45
Abbildung 22: Parameter einer PL/SQL-Anwendung	46
Abbildung 23: Methodenaufruf eines EJB-Objektes	47
Abbildung 24: Parameter einer EJB-Anwendung.....	48
Abbildung 25: OAS-Manager	51
Abbildung 26: OAS-Ebene im Navigationsbaum des OAS-Manager.....	52
Abbildung 27: HTTP-Listener-Ebene im Navigationsbaum des OAS-Manager	52
Abbildung 28: Anwendungsebene im Navigationsbaum des OAS-Manager.....	52

Abbildung 29: Anlegen eines DAD	54
Abbildung 30: Transaktionsfähiger DAD	55
Abbildung 31: Auswählen eines DAD für eine PL/SQL-Cartridge	55
Abbildung 32: Aufbau einer JDBC-Datenbankverbindung	56
Abbildung 33: Datenbankverbindung über JDBC zu einem DAD	57
Abbildung 34: PL/SQL-Package-Spezifikation und generierte Wrapper-Klasse	58
Abbildung 35: Aufruf einer PL/SQL-Funktion aus Java	59
Abbildung 36: Datenbankverbindung über ICX	60
Abbildung 37: ICX-Aufruf einer PL/SQL-Cartridge aus einer LiveHTML-Anwendung	61
Abbildung 38: URL-Format einer Anfrage an die ODBC-Cartridge	62
Abbildung 39: URL-Beispiel einer Anfrage an die ODBC-Cartridge	62
Abbildung 40: Anfrage/Antwort-Modell	63
Abbildung 41: Mehrere Anfragen in einer Sitzung	64
Abbildung 42: Aktivieren einer Sitzung im OAS-Manager	65
Abbildung 43: Banküberweisung als Transaktion	67
Abbildung 44: Komponenten einer Transaktion	68
Abbildung 45: Definieren einer web-deklarativen Transaktion	71
Abbildung 46: Deployment-Descriptor des EJB-Objektes "bank"	73
Abbildung 47: Deployment-Descriptor des EJB-Objektes "protokoll"	73
Abbildung 48: Transaktionsbereich einer Anwendung	74
Abbildung 49: Quellcode der Klassen "Bank" und "Protokoll"	74
Abbildung 50: Codefragment einer programmtechnischen Transaktion mit JTS	76
Abbildung 51: Auth-Broker und Auth-Provider	79
Abbildung 52: Definieren der Modi für Auth-Broker und Auth-Provider	81
Abbildung 53: Bereich, Gruppe und Benutzer bei Basic und Digest	82
Abbildung 54: Bereich, Gruppe und Benutzer bei Basic_Oracle	83
Abbildung 55: Authentifizierungsschemata im OAS-Manager	86
Abbildung 56: Zuordnung eines virtuellen Pfades zu einem Schema	86
Abbildung 57: Formular zur Angabe des Authentication-String	87
Abbildung 58: Authentication-String: Format und Beispiel	87
Abbildung 59: Sicherheit der Kommunikationswege	90
Abbildung 60: SSL-Formular eines Listeners	91

Abbildung 61: Installation des PL/SQL-WebToolkit	94
Abbildung 62: PL/SQL-Prozedur zur Generierung eines HTML-Dokumentes	95
Abbildung 63: URL-Format zum Aufrufen einer PL/SQL-Anwendung	97
Abbildung 64: URL zum Aufrufen der PL/SQL-Anwendung im Beispiel	97
Abbildung 65: Anzeige des HTML-Dokumentes im Browser	97
Abbildung 66: Parameterübergabe beim Aufruf einer PL/SQL-Prozedur.....	98
Abbildung 67: Perl-Skript zur Generierung eines HTML-Dokumentes.....	99
Abbildung 68: Anzeige des generierten HTML-Dokumentes	100
Abbildung 69: Template-File einer LiveHTML-Anwendung	101
Abbildung 70: Generiertes HTML-Dokument und dessen Anzeige	102
Abbildung 71: Anfrage an eine JServlet-Cartridge.....	105
Abbildung 72: Quellcode eines HTTP-Servlet	106
Abbildung 73: Anwendungstyp JServlet auswählen	107
Abbildung 74: Anwendungsnamen angeben	107
Abbildung 75: Cartridge hinzufügen	108
Abbildung 76: Stateful und stateless C++-Cartridge.....	110
Abbildung 77: Aufruf einer stateless C++-Cartridge.....	111
Abbildung 78: Aufruf einer stateful C++-Cartridge	112
Abbildung 79: IDL-File einer C++-Anwendung	114
Abbildung 80: Codefragment eines C++-Header-File.....	115
Abbildung 81: Codefragment eines C++-Body-File.....	115
Abbildung 82: Deployment-Descriptor-File einer C++-Anwendung.....	116
Abbildung 83: Codefragment des Client eines C++-Objekt.....	117
Abbildung 84: EJB-Architektur auf dem OAS	118
Abbildung 85: Zugriff eines Client auf eine Session-Bean	119
Abbildung 86: Quellcode einer Session-Bean-Klasse	121
Abbildung 87: Quellcode eines Home-Interface	122
Abbildung 88: Quellcode eines Remote-Interface.....	123
Abbildung 89: Quellcode zur Erstellung der Deployment-Descriptors.....	124
Abbildung 90: Initial Manifest-File	126
Abbildung 91: Anweisung zur Erstellung eines ejb-jar-File	126
Abbildung 92: Quellcode des Client einer EJB-Anwendung	127

Abbildung 93: Datenmodell	133
Abbildung 94: Anwendungsarchitektur	135
Abbildung 95: Maske "Veranstaltung auswählen"	138
Abbildung 96: Maske "Platzwahl"	139
Abbildung 97: Maske "Buchungsdaten erfassen"	140
Abbildung 98: Maske "Buchungsdaten erfassen" (Neukunde).....	141
Abbildung 99: E-Mail "Buchungsbestätigung"	142

Tabellenverzeichnis

Seite

Tabelle 1: Unterschiede zwischen Session-Bean und Entity-Bean	25
Tabelle 2: Listener und Default Ports eines Primary-Node	36
Tabelle 3: OAS-Komponenten.....	38
Tabelle 4: ORB-Komponenten	39
Tabelle 5: Datenbankzugriff der verschiedenen Anwendungstypen	53
Tabelle 6: Anfragearten an die ODBC-Cartridge	61
Tabelle 7: Definieren von Transaktionen bei den Anwendungstypen.....	70
Tabelle 8: Transaktionsmodi bei der deployment-deklarativen Transaktion.....	72
Tabelle 9: Beispiele für Host-Authentifizierungsschemata	85
Tabelle 10: Packages des PL/SQL-WebToolkit	94

1 Einleitung

Immer mehr Unternehmen nutzen heute Intranet und Internet zur Abwicklung ihrer Geschäftsprozesse. Dazu stellen diese Unternehmen ihre Anwendungen über das World Wide Web (WWW) oder Intranet ihren Kunden oder Mitarbeitern zur Verfügung.

Bei den bestehenden Anwendungen handelt es sich in vielen Fällen um individuelle Lösungen, die meistens nicht im WWW eingesetzt werden können. Es ist wünschenswert, die bestehenden und die neu entwickelten Anwendungen durch eine einheitliche Plattform zu integrieren und diese im WWW verfügbar zu machen. Zu diesem Zweck haben sich als Standard Application-Server etabliert, die mittlerweile von vielen Herstellern angeboten werden.

Doch was ist eigentlich ein Application-Server, wie ist ein Application-Server aufgebaut und welche Vorteile bietet sein Einsatz? Kurz gesagt ist ein Application-Server eine zentrale Plattform für den Einsatz von Anwendungen im WWW oder im Intranet. Der Application-Server stellt Middleware-Funktionen wie Skalierbarkeit, Zuverlässigkeit und Sicherheit bereit, ohne dass der Entwickler diese Aspekte bei der Anwendungsentwicklung explizit berücksichtigen muss.

Die vorliegende Diplomarbeit beschäftigt sich ausführlich mit dem Thema Application-Server. Dabei wird speziell auf den Oracle Application-Server (OAS) eingegangen. Dem Leser soll die Architektur des OAS verständlich gemacht werden, sowie ein Überblick über die Einsatzmöglichkeiten des OAS gegeben werden.

Für die ausführliche Darstellung der einzelnen Themen ist die vorliegende Diplomarbeit in drei Teile eingeteilt.

Im ersten Teil dieser Diplomarbeit erhält der Leser eine Einführung in die grundlegenden Begriffe und Technologien, die notwendig sind, um die Funktionsweise eines Application-Server im WWW oder im Intranet verstehen zu können. Dazu werden die technologischen Grundlagen des Internet wie TCP/IP und World Wide Web behandelt. Es folgt ein

Einblick in die verschiedenen Netzwerkarchitekturen. Abgeschlossen wird dieser Teil mit einem kurzen Überblick über die Komponentenmodelle CORBA und Enterprise JavaBeans.

Im zweiten Teil dieser Diplomarbeit wird der Oracle Application-Server in der Version 4.0.8.1 vorgestellt. Nach einer Einführung in das Thema Application-Server wird die Architektur des OAS beschrieben. Daran anschließend werden die Middleware-Funktionen des OAS behandelt, wie Skalierbarkeit, Datenbankzugriff, Interaktionsmodelle und Sicherheitsaspekte. Zum Abschluss dieses Teils werden dem Leser anhand von Beispielen die Einsatzmöglichkeiten des OAS dargestellt, also welche Arten von Anwendungen auf dem OAS eingesetzt werden können.

Im dritten Teil wird die Anwendung "OnlineTicket" beschrieben, die für diese Diplomarbeit entwickelt wurde. Diese Anwendung unterstützt den Vertrieb von Tickets für Veranstaltungen.

Im Anhang findet sich ein Glossar, in dem die wichtigsten Fachbegriffe dieser Diplomarbeit übersichtlich zusammengefasst und kurz erklärt sind. Zusätzlich ist der gesamte Quellcode der Anwendung im Anhang zu finden.

2 Grundlagen

In diesem Kapitel werden die Grundlagen der Kommunikation im Internet dargestellt, soweit sie für das Verständnis des OAS relevant sind. Im wesentlichen werden die Protokolle TCP/IP und die Technologien des World Wide Web (WWW) behandelt. Weiterhin werden die verschiedenen Netzwerkarchitekturen vorgestellt und ihre Vorteile und Nachteile erläutert. Zuletzt folgt eine Betrachtung der beiden Architekturen verteilter objektorientierter Anwendungen, CORBA und Enterprise JavaBeans.

Prozess¹

Da im Folgenden der Begriff "Prozess" benutzt wird, soll hier kurz definiert werden was unter einem Prozess zu verstehen ist:

Def.: Prozess = Eine Anwendung während der Ausführung.

Wird eine Anwendung aufgerufen, so wird der zugehörige Code in den Hauptspeicher geladen und dann ausgeführt. Diese ausgeführte Anwendung wird als Prozess bezeichnet. Wird die gleiche Anwendung gleichzeitig von mehreren Benutzern gestartet, so handelt es sich dabei um zwei verschiedene Prozesse, obwohl beide die gleiche Anwendung ausführen.

Schichten-Modell

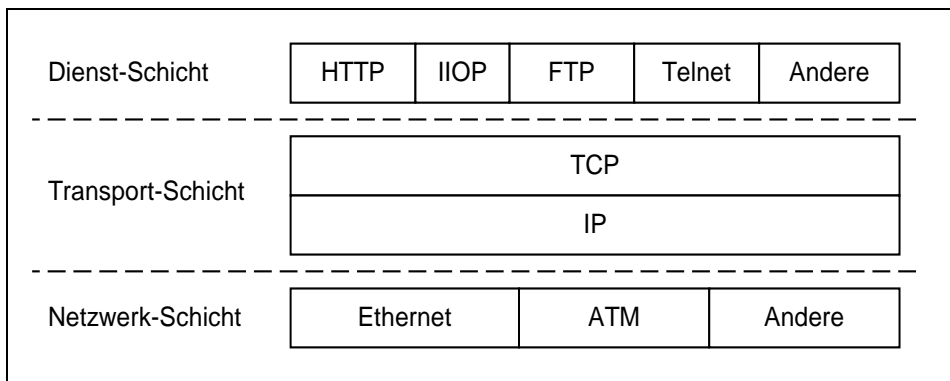
Eines der grundsätzlichen Prinzipien der Rechner-Kommunikation ist die Schichtung der Teilfunktionen einer Kommunikation. Die Schichtung dieser Teilfunktionen wird mit Hilfe eines Schichten-Modells veranschaulicht.

In Abbildung 1 wird dargestellt, wie das Schichten-Modell bei der Rechner-Kommunikation im Internet über TCP/IP² aufgebaut ist.

¹ Herold, Helmut: UNIX-Grundlagen: Kommandos und Konzepte. Bonn 1994, S. 476.

² Siehe Kapitel "2.1 TCP/IP".

Abbildung 1: Schichten der Rechner-Kommunikation über TCP/IP



Quelle: In Anlehnung an: Wilde, Erik: World Wide Web. Technische Grundlagen. Berlin 1999, S. 26.

Zum Verständnis der Rechner-Kommunikation im Internet reicht dieses drei-schichtige Modell völlig aus. Es ist eine Vereinfachung des OSI-Referenzmodells der ISO, das aus sieben Schichten besteht.

Der jeweilige Dienst der Dienst-Schicht nimmt die Daten der Anwendung entgegen und leitet sie an die Transport-Schicht weiter. Die Transport-Schicht überträgt die Daten nach bestimmten Regeln, die durch ihre Protokolle definiert sind, über das Netzwerk. Die unterste Schicht, die Netzwerk-Schicht, repräsentiert das tatsächlich existierende Netzwerk, das die physikalische Infrastruktur für die Übertragung der Informationen bereitstellt.

2.1 TCP/IP

Internet Protocol

Das Internet Protocol (IP) ist ein Protokoll zur Übertragung von Informationen, die in Pakete verpackt sind.

Um die Kommunikation der Rechner innerhalb des Internets überhaupt zu ermöglichen, muss jeder Rechner eine Adresse haben, die innerhalb des gesamten Internet eindeutig ist. Diese Adresse wird als IP-Adresse bezeichnet. Die IP-Adresse ist eine 32-Bit-Zahl und

wird in 4 Bytes unterteilt, wobei die einzelnen Bytes jeweils durch einen Punkt voneinander getrennt sind (z. B.: 192.168.1.160).

Bei der Übertragung mit dem IP werden alle zu übertragenden Informationen in Pakete verpackt. Der sendende Rechner schickt seine Pakete ins Netz und die Router sorgen dafür, dass die Pakete beim empfangenden Rechner ankommen. Die Pakete werden im Internet über vorher nicht festgelegte Transportwege übertragen, wobei die Transportwege der einzelnen Pakete einer Sendung unterschiedlich sein können. Bedingt durch diese unterschiedlichen Transportwege können die Pakete einer Sendung in einer anderen Reihenfolge beim empfangenden Rechner ankommen, als sie beim sendenden Rechner abgeschickt wurden. Beim empfangenden Rechner müssen die Pakete dann wieder in der richtigen Reihenfolge zusammengesetzt werden.

Ein solches Paket heißt IP-Paket, da es mit dem IP übertragen wird. Ein IP-Paket wird auch als Datagramm bezeichnet. Das IP-Paket ist aus 2 Teilen aufgebaut, dem IP-Header und den IP-Nutzdaten. Der IP-Header enthält neben anderen Informationen wichtige Angaben, wie die Nummer des IP-Paketes, die Nummer des nachfolgenden IP-Paketes, eine Prüfsumme, die IP-Adresse des sendenden Rechners, sowie die IP-Adresse des empfangenden Rechners.

Bei der Übertragung der IP-Pakete mit dem IP gibt es keine Mechanismen zur Kontrolle des Austausches. Diese Mechanismen, wie eine Kontrolle der empfangenen Pakete und eine eventuelle Neuansforderung eines nicht empfangenen Paket, werden auf der Ebene des TCP realisiert.

Transmission Control Protocol

Das Transmission Control Protocol (TCP) setzt auf dem IP auf. Das TCP ist ein Verfahren, das beim Verbindungsauf- und -abbau, sowie beim Datenaustausch der IP-Pakete genutzt wird. TCP fügt an die zu sendenden Daten einen TCP-Header hinzu und gibt das Paket als IP-Nutzdaten an das IP weiter. Das IP fügt am Anfang einen IP-Header hinzu und schickt dieses IP-Paket zur Übertragung ins Netz. Der empfangende Rechner kann aufgrund der Informationen im TCP-Header feststellen, ob alle gesendeten Pakete empfangen wurden und bei Bedarf ein nicht empfangenes Paket neu anfordern.

Port

Damit sich bei parallelen Verbindungen zwischen zwei Rechnern die empfangenen Pakete nicht vermischen, gibt es bei einer Kommunikation über TCP/IP sogenannte Ports. Diese Ports sind keine physischen Anschlüsse an den Rechnern, sondern Nummern mit einer Länge von 16 Bit, die vom sendenden Rechner in den TCP-Header eingetragen werden. Der empfangende Rechner lauscht auf Port-Nummern zwischen 1 und 65535 im TCP-Header. Entdeckt er eine Port-Nummer, kann er aufgrund dieser das Paket der richtigen Anwendung auf dem empfangenden Rechner zuordnen. Für die Standard-Internet-Dienste gibt es Default-Ports, wie z. B. 80 für HTTP. Das heißt die Angabe eines Ports ist optional, wenn ein Default-Port angesprochen werden soll. Die Kombination einer IP-Adresse mit einer Port-Nummer wird Socket genannt.

2.2 World Wide Web

Das World Wide Web (WWW) ist der bekannteste und am weitesten verbreitete Internet-Dienst. Es besteht aus einer Menge von Anwendungen, die auf der vom Internet bereitgestellten Infrastruktur ausgeführt werden.

Das WWW basiert auf dem Konzept des Hypertext, in dem Dokumente als miteinander verbundene Informationsteile modelliert werden. Da diese Dokumente aber nicht nur aus Text, sondern auch aus Graphiken, sowie aus Video- und Audiosequenzen (Multimedia) bestehen können, hat sich der Begriff Hypermedia etabliert, der die Kombination von Hypertext und Multimedia besser ausdrückt.

Die Technologie des World Wide Web basiert im wesentlichen auf den folgenden drei Komponenten:

- (1) URI / URL
- (2) HTTP
- (3) HTML / XML

(1) URI / URL

Universal Resource Identifier (URI) ist die Bezeichnung für alle Typen von Namen und Adressen, die auf Objekte im WWW verweisen.

Durch einen Uniform Resource Locator (URL) wird ein Dokument eindeutig im WWW identifiziert. Die URL ist dabei nur ein Typ eines URI. In Abbildung 2 wird gezeigt wie eine URL aufgebaut ist.

Abbildung 2: Aufbau einer URL

```
schema://host:port/path
```

Diese Notation wird anhand eines Beispiels verdeutlicht. In Abbildung 3 wird eine URL dargestellt.

Abbildung 3: Beispiel einer URL

```
http://pc076.nochen.opitz-partner.de:80/test/HelloWorld
```

Die Angabe "http" ist das Schema und kennzeichnet das zu verwendende Protokoll. Die Angabe des Host "pc076.nochen.opitz-partner.de" wird als Domain-Name bezeichnet. Diese Angabe wird von einem DNS (Domain Name Server) aufgelöst, der seinerseits die entsprechende IP-Adresse zurückgibt, die den gewünschten Host adressiert. Die Angabe "80" kennzeichnet den zu verwendenden Port. Der Pfad "/test/HelloWorld" gibt an, wie das angefragte Dokument heißt und wo es auf dem angegebenen Host zu finden ist. Es handelt sich hierbei um die Angabe eines virtuellen Pfades und nicht um die Angabe eines physikalischen Pfades. Die Zuordnung zwischen virtuellem Pfad und physikalischem Pfad ist auf dem Host hinterlegt.

(2) HTTP

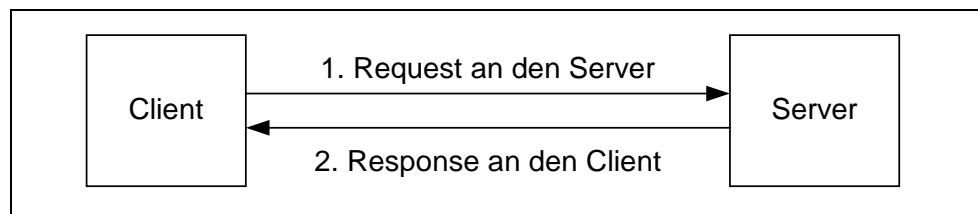
Das Hypertext Transfer Protocol (HTTP) ist ein Anfrage/Antwort-Übertragungsprotokoll im WWW, zur Übertragung nicht lokal gespeicherter Informationen. Die zur Zeit aktuelle Version ist HTTP/1.1. Es basiert auf einer Client/Server-Architektur, bei der ein Client mit

einem Web-Server Kontakt aufnimmt und eine Anfrage (Request) an diesen absendet. Der Web-Server bearbeitet die Anfrage und sendet eine Antwort (Response) zurück an den Client. Die Kommunikation über HTTP zwischen Client und Server läuft folgendermaßen ab:

1. Der Client baut die Verbindung zum Server auf (IP-Connect).
2. Der Client sendet seine Anfrage in Form einer URL zum Server. Die URL verweist auf das vom Benutzer gewünschte Dokument.
3. Der Server findet das gewünschte Dokument auf, bzw. generiert es und sendet es an den Client.
4. Der Server schließt die Verbindung zum Client.

In Abbildung 4 wird der Ablauf der Kommunikation über HTTP zwischen Client und Server veranschaulicht.

Abbildung 4: Kommunikation über HTTP zwischen Client und Server



Diese Kommunikation über HTTP wird auch als HTTP-Interaktion oder Anfrage/Antwort-Interaktion (Request/Response-Interaction) bezeichnet.

Beim HTTP handelt es sich um ein zustandsloses Protokoll. Das heißt jede HTTP-Interaktion ist genau eine Sitzung (Session). Wünschenswert ist, dass eine Sitzung aus mehreren HTTP-Interaktionen besteht. Dazu ist aber eine Information über den aktuellen Zustand der Sitzung notwendig. Diese Information kann mit Hilfe eines Cookies realisiert werden. Im Grunde genommen handelt es sich bei einem Cookie um eine zwischen Client und Server ausgetauschte Information, die zum Aufrechterhalten der Information über den Zustand dient¹. Auf diese Weise wird eine logische Sitzung hergestellt, die keinen Bezug

¹ Vgl. Wilde, Erik: World Wide Web. Berlin 1999, S. 135

zu einer physikalischen Entsprechung aufweist, wie beispielsweise einer persistenten Verbindung. Die Cookies werden server-seitig in den Header des HTML-Dokumentes eingebettet, das an den Client übertragen wird. Die Cookies werden client-seitig gespeichert und bleiben bis zu einem vom Server angegebenen Verfallsdatum gültig.

(3) HTML / XML

Die Hypertext Markup Language (HTML) ist eine Beschreibungssprache im ASCII-Format zur Beschreibung von HTML-Dokumenten. Die momentan aktuelle HTML-Version ist 4.0.

HTML-Dokumente können durch einen HTML-Browser interpretiert und dargestellt werden. Dabei sind zwischen dem darzustellenden Inhalt eines HTML-Dokumentes sogenannte Tags notiert, die für das Strukturieren der Daten zum Zeitpunkt der Anzeige zuständig sind.

Im Gegensatz zu HTML ist die eXtensible Markup Language (XML) eigentlich gar keine Sprache, die direkt die Anzeige beeinflusst, sondern mehr eine Sprache mit der eine andere Sprache beschrieben werden kann. Diese Art Sprache wird auch als Meta-Sprache bezeichnet. Es wird nur festgelegt, wie Tags auszusehen haben, die Bedeutung der Tags aber wird durch das sogenannte DTD (Document Type Definition) -File festgelegt.

2.3 Netzwerkarchitekturen

Werden Anwendungen von mehreren Benutzern verwendet, so stellt man sie über ein Netzwerk zur Verfügung. Allgemein besteht jede Anwendung aus drei logischen Komponenten:

- **Präsentation**

Die Präsentation (Presentation Logic) umfasst dabei alle Funktionen die notwendig sind, damit die Verarbeitungsleistung der Anwendung dem Benutzer sichtbar wird.

- **Anwendungslogik**

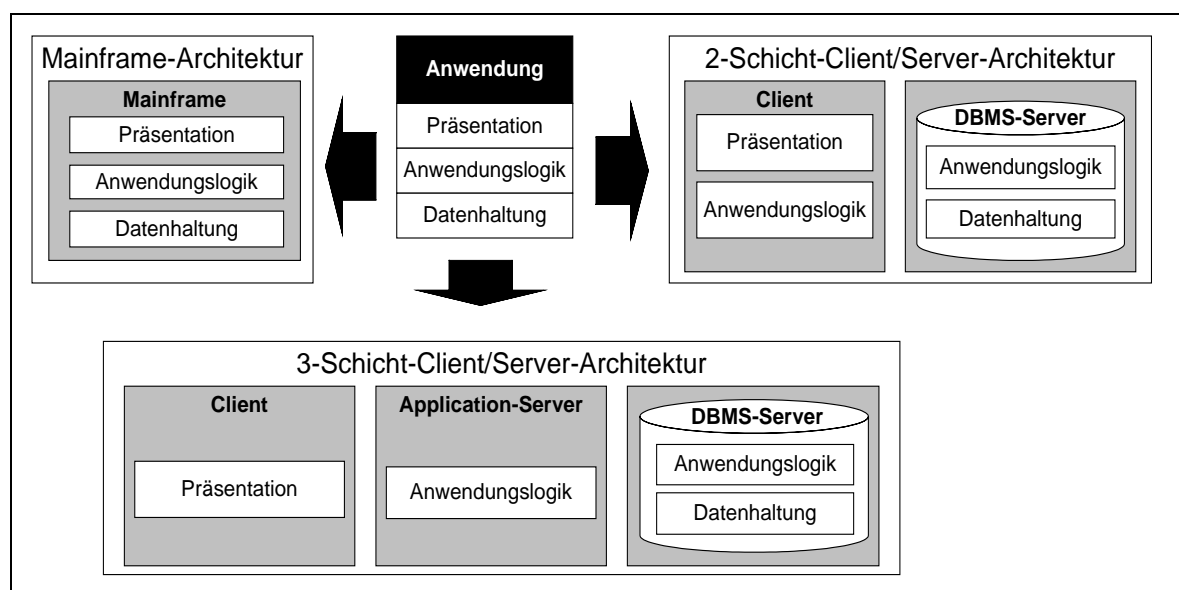
Hinter der Anwendungslogik (Business Logic) verbirgt sich die eigentliche Verarbeitungslogik der Anwendung.

- Datenhaltung

Die Datenhaltung (Data Storage) bietet alle Funktionen, damit Daten sich persistent verwalten lassen, z. B. mit einem relationalen Datenbank Management System (RDBMS).

Diese Komponenten lassen sich verschieden anordnen, so dass sich drei Netzwerkarchitekturen ergeben, die in Abbildung 5 als Übersicht dargestellt werden.

Abbildung 5: Anordnungen der Komponenten einer Anwendung



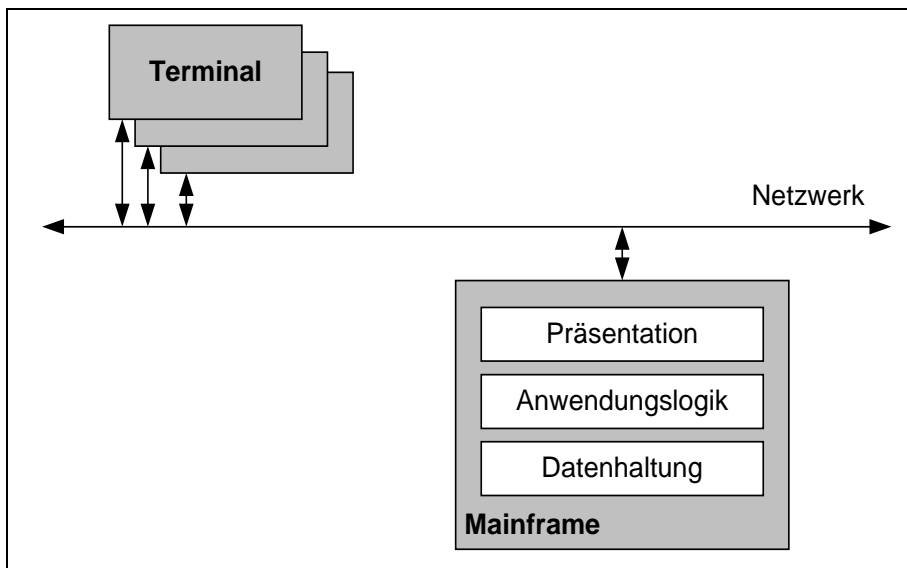
Mainframe-Architektur

Ein Mainframe ist ein Großrechner, auf dem alle Prozesse laufen. Als Ein-/Ausgabe-Schnittstelle für den Benutzer dienen Terminals. Die Benutzer melden sich über die Terminals am Mainframe an.

In Abbildung 6 wird die Mainframe-Architektur gezeigt. Hierbei sind alle Komponenten auf einer Schicht angeordnet.

Die Mainframe-Architektur hat den wesentlichen Vorteil, dass das Anwendungsmanagement einfach ist. Änderungen an Hardware und Software müssen nur an einer zentralen Stelle vorgenommen werden.

Abbildung 6: Mainframe-Architektur



Bei dieser Architektur ist nachteilig, dass die Hardware sehr teuer ist. Zudem wird meist durch die große Anzahl von Anwendungen und Benutzern eine regelmäßige Auslastung des Mainframes erreicht. Skalierbarkeit von Mainframes ist schwierig zu realisieren und mit hohen Kosten verbunden, da neue Hardware angeschafft werden muss.

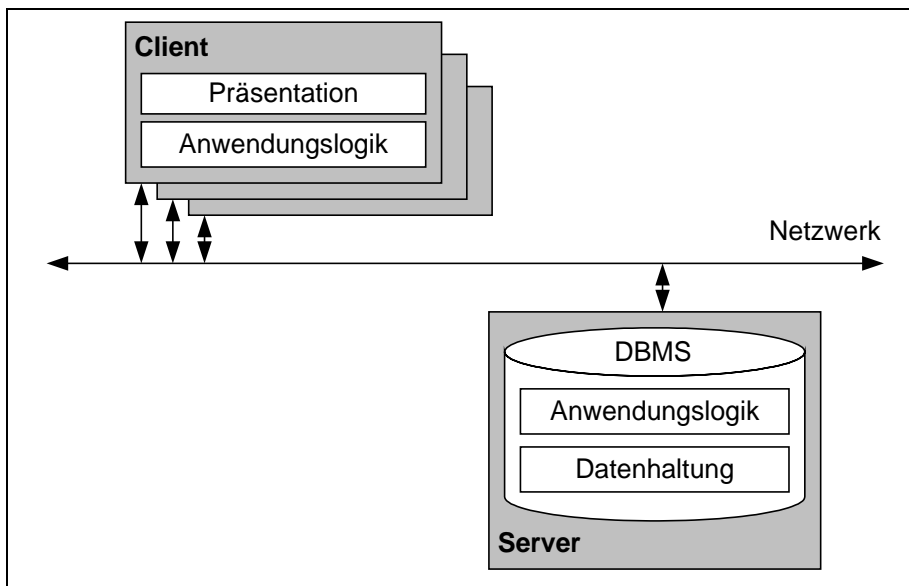
2-Schicht-Client/Server-Architektur

Das einfachste Modell einer Client/Server-Architektur ist die 2-Schicht-Client/Server-Architektur. In Abbildung 7 wird die 2-Schicht-Client/Server-Architektur dargestellt.

In diesem Modell wird die 1. Schicht durch den Server gebildet. Der Server kann ein DBMS (Datenbank Management System) -Server sein. Dieser ist für die Datenhaltung und den daten-intensiven Teil der Anwendungslogik zuständig, der in der Datenbank gespeichert ist (Stored-Procedures). Für den Betrieb im WWW kann der Server auch ein Web-Server sein.

Die 2. Schicht bilden die Clients, die als Graphical-User-Interfaces (GUIs) dienen. Die Clients führen die Präsentation, sowie den rechen-intensiven Teil der Anwendungslogik aus. Da der rechen-intensive Teil der Anwendungslogik sehr komplex sein kann, müssen die Client-Rechner sehr leistungsfähig sein und werden deshalb "Fat-Clients" genannt.

Abbildung 7: 2-Schicht-Client/Server-Architektur



Von Vorteil bei einer 2-Schicht-Client/Server-Architektur ist, dass ein Teil der Rechenleistung, die bei der Mainframe-Architektur vom Mainframe verlangt würde, auf die Clients verlagert wird. Aus diesem Grund sind kostengünstigere Server (PCs) ausreichend, statt eines teuren Mainframes.

Skalierbarkeit ist gegenüber der Mainframe-Architektur durch den Einsatz mehrerer Server kostengünstig realisierbar.

Von Nachteil bei einer 2-Schicht-Client/Server-Architektur sind die hohen Kosten, für Einbau und Austausch von Hardware, sowie Installation und Wartung von Software auf den Clients, da dies auf allen Clients durchgeführt werden muss.

Zudem bestehen Sicherheitsbedenken beim Einsatz dieser Architektur im WWW, da DBMS-Server und Web-Server auf einem Rechner ausgeführt werden, und somit die Datenbank direkt aus dem Internet sichtbar ist.

Zusätzlich gibt es keine klare Trennung zwischen der Präsentation und der Anwendungslogik. Bei einer Änderung an der Präsentation oder der Anwendungslogik sind aufwendige Codeänderungen notwendig, da beide miteinander vermischt sind.

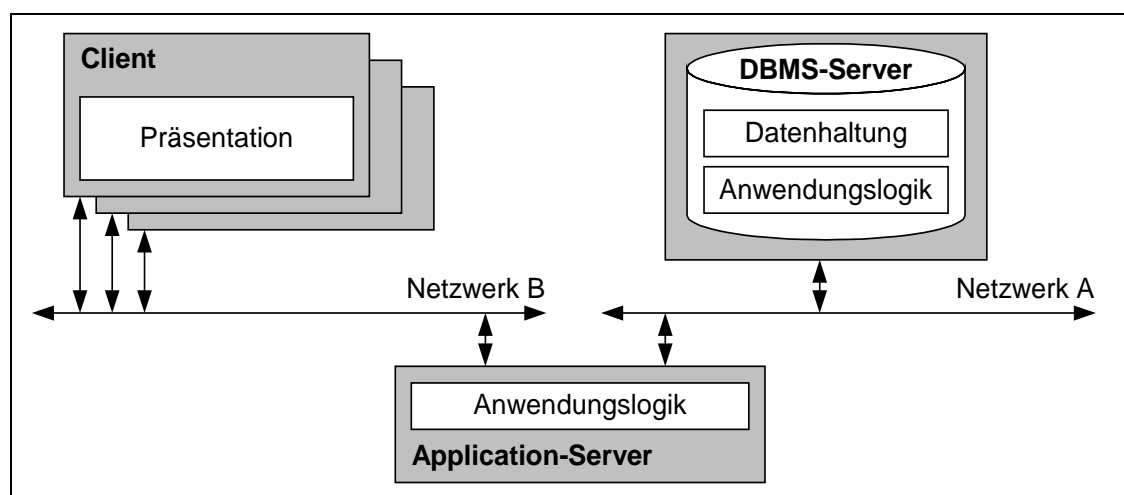
Beim Datenaustausch zwischen Clients und Server entsteht ein hohes Transportaufkommen im Netzwerk, da die Daten zentral auf dem Server gehalten werden, aber dezentral auf den Clients verarbeitet werden.

3-Schicht-Client/Server-Architektur

Bei einer 3-Schicht-Client/Server-Architektur wird zwischen den Clients und dem Server eine weitere Schicht implementiert. Diese Schicht wird Mittel-Schicht oder auch Application-Server-Schicht genannt. Der Client und der DBMS-Server kommunizieren hier nicht direkt miteinander. In Abbildung 8 wird die 3-Schicht-Client/Server-Architektur dargestellt.

In diesem Modell wird die 1. Schicht durch den DBMS-Server gebildet, auf dem das DBMS ausgeführt wird. Das DBMS ist für die Datenhaltung, sowie den daten-intensiven Teil der Anwendungslogik zuständig, der in der Datenbank gespeichert ist (Stored-Procedures).

Abbildung 8: 3-Schicht-Client/Server-Architektur



Die 2. Schicht, die Mittel-Schicht, implementiert die rechen-intensive Anwendungslogik. Auf dieser Mittel-Schicht ist meistens ein Application-Server¹ implementiert, der im wesentlichen die Aufgabe hat Anwendungen zur Verfügung zu stellen.

Die 3. Schicht bilden die Clients, die als Graphical-User-Interface dienen und nur die Präsentation ausführen. Da bei dieser Architektur keine Anwendungslogik auf den Client-Rechnern ausgeführt wird, reichen Rechner mit geringer Leistungsfähigkeit aus. Die

¹ Siehe Kapitel "3.1 Einführung".

Client-Rechner werden nur noch mit einem Web-Browser und optional mit einer Java Virtual Machine (JVM) ausgestattet und deshalb "Thin-Clients" genannt.

Aus Sicht eines Client hat der Application-Server zusammen mit dem DBMS-Server die Rolle eines Mainframe. Aus Sicht des DBMS-Server hat der Application-Server zusammen mit einem Client die Rolle eines Fat-Client. Die 3-Schicht-Client/Server-Architektur kombiniert also die Vorteile der Mainframe-Architektur mit den Vorteilen der 2-Schicht-Client/Server-Architektur.

Ein weiterer Vorteil der 3-Schicht-Client/Server-Architektur ist, dass mit Hilfe des Application-Server Skalierbarkeit und Sicherheit realisiert werden können.

Die gesamte Anwendungslogik wird zentral auf der Mittel-Schicht und in der Datenbank gehalten, woraus eine einfache und kostengünstige Wartung resultiert. Dadurch gibt es eine klare Trennung zwischen Präsentation und Anwendungslogik, so dass beide leicht zu ändern sind.

Es ergibt sich eine höhere Sicherheit beim Einsatz dieser Architektur im WWW, da DBMS-Server und Web-Server auf zwei verschiedenen Rechnern ausgeführt werden und somit die Datenbank nicht direkt aus dem Internet sichtbar ist.

Zusätzlich zu diesen drei grundlegenden Netzwerkarchitekturen gibt es N-Schicht-Client/Server-Architekturen. Hierbei wird die Mittel-Schicht in weitere Schichten aufgeteilt.

2.4 CORBA

Die Common Object Request Broker Architecture (CORBA) ist ein Industriestandard, der die Architektur verteilter objektorientierter Anwendungen vereinheitlicht und unabhängig ist von Rechnerarchitektur, Betriebssystem und Programmiersprache¹.

Dieser Standard wurde von der Object Management Group (OMG)² verfasst und liegt zur Zeit in der Version 2.3.1 vor.

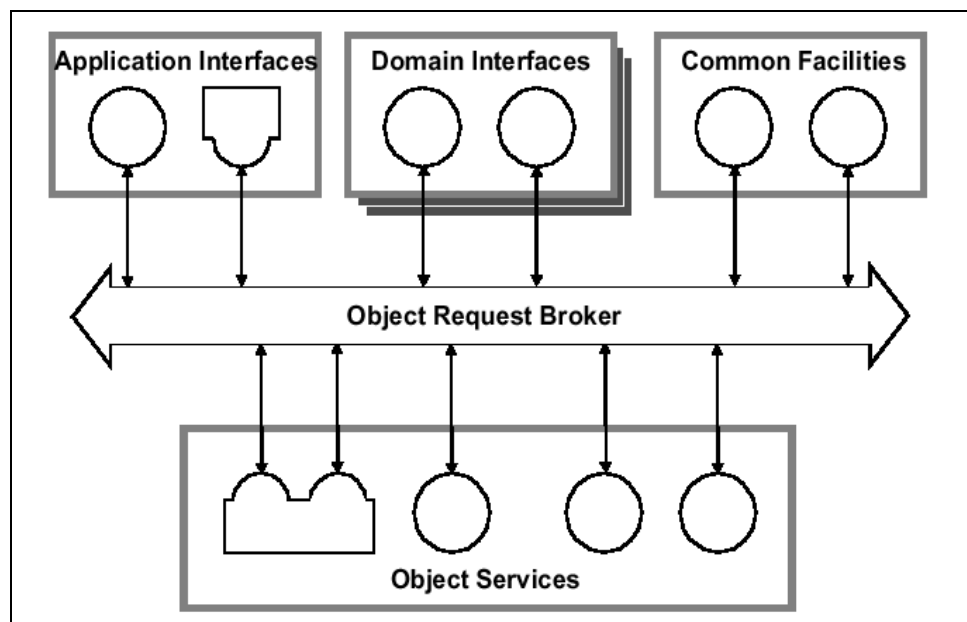
¹ Sayegh, Andreas: CORBA - Standard, Spezifikation, Entwicklung. Köln 1999, S. 9.

² Siehe Glossar im Anhang.

Object Management Architecture

Die konzeptionelle Infrastruktur auf der CORBA, sowie alle anderen Spezifikationen der OMG entwickelt wurden und weiterentwickelt werden, bildet eine abstrakte Beschreibung der Objektwelt, wie sie die OMG sieht. Diese wird in der Object Management Architecture (OMA) definiert. In Abbildung 9 wird das OMA-Referenzmodell gezeigt.

Abbildung 9: OMA-Referenzmodell



Quelle: The Common Object Request Broker: Architecture and Spezifikation, Revision 2.3.1. Oktober 1999, S. xxviii.

Das OMA-Referenzmodell ist der Schlüssel, um die Struktur von CORBA verstehen zu können. Es enthält die folgenden Komponenten:

- (1) Object Request Broker
- (2) Object Services
- (3) Common Facilities
- (4) Application Interfaces
- (5) Domain Interfaces

(1) Object Request Broker

Der Object Request Broker (ORB) ist der Kern der CORBA-Kommunikation. Er ermöglicht das Senden und Empfangen von Anfragen und Antworten zwischen den verteilten Objekten. Die Kommunikation ist dabei unabhängig davon, wie die Objekte implementiert sind. Die Gesamtheit aller Objekte, auf die ein ORB Zugriff hat, heißt ORB-Domäne.

(2) Object Services

Die Object Services sind eine Menge von Diensten (Schnittstellen und Objekte), die Basisfunktionalitäten für die Nutzung und Implementierung der Objekte unterstützen. Diese Dienste sind notwendig zur Konstruktion jeder verteilten Anwendung und sind immer unabhängig vom Anwendungsbereich. Die Object Services werden CORBAServices¹ genannt.

(3) Common Facilities

Common Facilities sind eine Menge von Diensten, die von mehreren Anwendungen gemeinsam genutzt werden können. Diese Dienste sind nicht so grundlegend wie die Object Services. Die Common Facilities werden "CORBAfacilities" genannt.

(4) Application Interfaces

Application Interfaces bezeichnen die Schnittstellen, die von einer Anwendung angeboten werden.

(5) Domain Interfaces

Domain Interfaces enthalten branchenspezifische Schnittstellen. Die OMG erwähnt Finanz- und Gesundheitswesen, sowie Telekommunikation².

¹ Siehe Abschnitt "CORBAServices".

² Sayegh, Andreas: CORBA - Standard, Spezifikation, Entwicklung. Köln 1999, S. 15.

Entferntes Objekt und Proxy

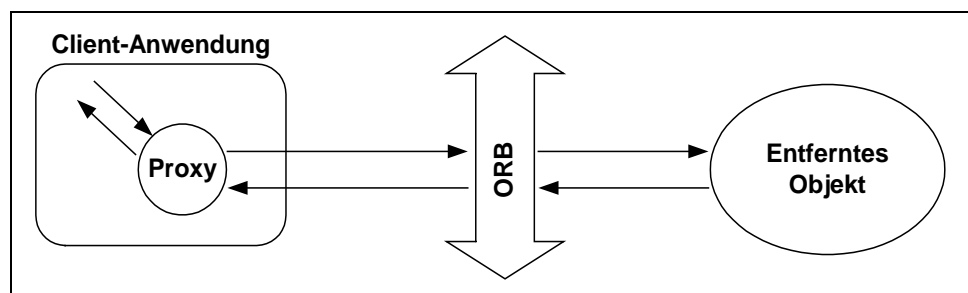
Da CORBA mit verteilten Objekten arbeitet, soll zunächst der Begriff des "entfernten Objektes" (remote object) definiert werden. Ein entferntes Objekt ist nicht auf dem lokalen Rechner gespeichert, sondern auf einen entfernten Rechner, zu dem eine physikalische Verbindung besteht. Zusätzlich müssen sich die Rechner auf einer logischen Ebene verständigen können, also die gleiche Sprache sprechen (z. B. CORBA).

Ein entferntes Objekt unterscheidet sich von einem nahen Objekt durch die Art des Zugriffs. Während ein nahes Objekt im Speicher des lokalen Rechners residiert und dort direkt manipuliert werden kann, so residiert ein entferntes Objekt im Speicher eines entfernten Rechners und kann dort nicht direkt manipuliert werden.

CORBA ist ein Client/Server-Modell. Um eine Methode eines entfernten Objektes aufzurufen, muss der Client eine Anfrage an den Server senden und eine Antwort darauf erhalten. Das Ausführen der Methode und die Rückgabe einer Antwort eines entfernten Objektes kann demnach als Dienst bezeichnet werden.

Der Client benötigt kein Wissen über den Ort, an dem sich das entfernte Objekt befindet. Er kann die Operation aufrufen, als wäre es ein lokales Objekt. Der entfernte Zugriff soll für den Client also transparent sein. Um die Transparenz umzusetzen, benötigt jedes entfernte Objekt ein nahes Objekt, das die gleiche Signatur besitzt wie das entfernte Objekt. Dieses nahe Objekt wird als Proxy-Objekt (proxy object) bezeichnet und hat die Aufgabe eine Anfrage an ein entferntes Objekt weiterzuleiten und dessen Antwort an den Client zurückzugeben. In Abbildung 10 wird ein Modell dieser Kommunikation dargestellt.

Abbildung 10: Schema der Proxy-Kommunikation



Quelle: Sayegh, Andreas: CORBA - Standard, Spezifikation, Entwicklung. 2. Aufl. Köln 1999, S. 9.

Das Proxy-Objekt konstruiert eine Anfrage, die mindestens den Operationsnamen, sowie alle Eingabeparameter enthält. Diese Nachricht wird an das entfernte Objekt gesendet. Das entfernte Objekt gibt die Antwort, die mindestens den Rückgabewert, sowie alle Ausgabeparameter enthält, zurück an das Proxy-Objekt. Das Proxy-Objekt reicht diese Daten an die Anwendung weiter.

Ein Proxy-Objekt ist eine Instanz einer Proxy-Klasse. Diese Proxy-Klasse wird in CORBA "Stub" genannt. Die Erstellung von Proxy-Klassen kann durch einen Generator automatisiert werden.

Zusätzlich zum Stub erzeugt der Generator einen "Skeleton". Der Skeleton ist ein Code-Skelett und stellt das server-seitige Gegenstück zum Stub dar. Er stellt die Schnittstelle für jede vom entfernten Objekt angebotene Methode bereit und leitet die Methodenaufrufe an das entfernte Objekt weiter. Für das Generieren von Stub und Skeleton müssen jedoch die zu realisierenden Objektschnittstellen bekannt sein.

Deshalb gibt es in CORBA die von der OMG definierte Interface Definition Language (IDL). Die IDL ermöglicht die einheitliche, formale und implementierungsunabhängige Beschreibung eines entfernten Objektes. Sie beschreibt die Attribute und Operationen mit deren Parametern, die von dem entfernten Objekt angeboten werden und von den Clients aufgerufen werden können.

Da Stubs und Skeletons aus diesen IDL-Definitionen generiert werden, wird der Generator auch als IDL-Compiler bezeichnet.

Entfernte Objekte müssen erzeugt werden, bevor der Zugriff auf sie möglich ist und gelöscht werden, wenn sie nicht mehr benötigt werden. Diese Vorgänge gehören zur Steuerung des Lebenszyklus (life cycle) eines Objektes und werden von einem Standard-Dienst angeboten, dem Life-Cycle-Service¹.

Struktur eines ORB

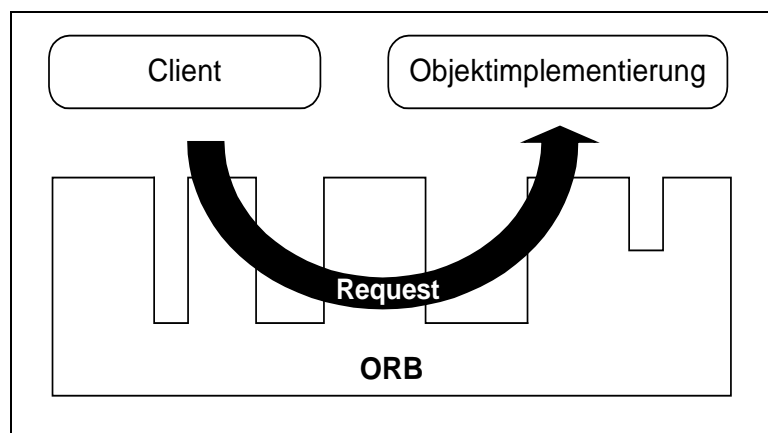
Der ORB ist verantwortlich für alle Mechanismen, die benötigt werden um eine Objektimplementierung zu finden, an die sich eine Anfrage richtet, diese vorzubereiten für

¹ Siehe CORBAServices.

den Empfang der Anfrage und die Antwort der Objektimplementierung an den Client zurückzugeben. Die Schnittstelle, die der Client sieht, ist unabhängig davon, wo die Objektimplementierung sich befindet, in welcher Programmiersprache sie implementiert ist, oder von sonstigen Aspekten, die nicht durch die Schnittstelle der Objektimplementierung wiedergespiegelt werden.

In Abbildung 11 wird eine Anfrage von einem Client an eine Objektimplementierung gesendet.

Abbildung 11: Anfrage über den ORB



Quelle: The Common Object Request Broker: Architecture and Spezifikation, Revision 2.3.1. Oktober 1999, S. 2-2.

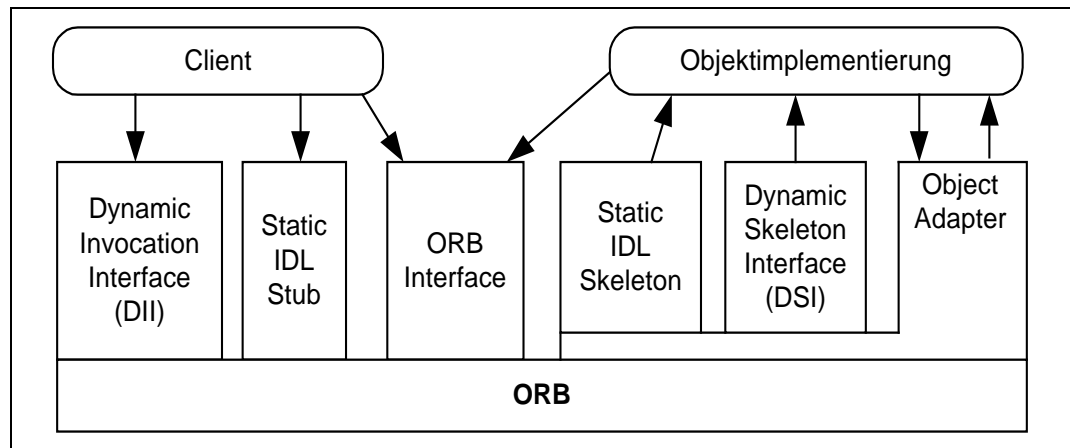
Die Struktur eines ORB bietet verschiedene Schnittstellen, die für Anfragen der Clients über einen ORB an eine Objektimplementierung benutzt werden können. Grundsätzlich kann die Schnittstelle zu einem Objekt auf zwei Arten definiert werden.

Die erste Möglichkeit ist, die Schnittstelle statisch in einer Interface Definition Language (IDL) zu definieren. Diese Sprache definiert die Objekte laut ihren Operationen und deren Parameter.

Die zweite (alternative oder zusätzliche) Möglichkeit ist, die Schnittstelle einem Interface-Repository hinzuzufügen. Das Interface-Repository ist ein Dienst, der die Komponenten einer Schnittstelle als Objekte darstellt und den Zugriff auf diese Komponenten zur Laufzeit erlaubt. In jeder ORB-Implementierung sind IDL-Definition und Interface-Repository gleichwertige Möglichkeiten um Schnittstellen zu definieren.

In Abbildung 12 wird dargestellt wie die Struktur eines ORB diese beiden Möglichkeiten unterstützt.

Abbildung 12: Struktur eines ORB



Quelle: In Anlehnung an: The Common Object Request Broker: Architecture and Spezifikation, Revision 2.3.1. Oktober 1999, S. 2-3.

Der Client kann für seine Anfrage das "Dynamic Invocation Interface (DII)" oder einen "Static IDL Stub" benutzen. Das DII ist für alle Anfragen identisch und unabhängig von der Schnittstelle des Zielobjektes (dynamisch definierte Schnittstelle). Der Static IDL Stub ist für jedes Zielobjekt spezifisch und somit abhängig von der Schnittstelle des Zielobjektes (statisch definierte Schnittstelle).

Die Objektkomponentierung empfängt die Anfrage entweder über den "Static IDL Skeleton" oder über das "Dynamic Skeleton Interface (DSI)".

Der "Object Adapter" bietet den grundsätzlichen Zugriff für die Objektkomponentierung auf Dienste, die der ORB anbietet. Dienste eines ORB, die durch den Object Adapter angeboten werden, sind meistens: Generieren und Interpretieren von Objektreferenzen, Methodenaufruf, Sicherheit der Kommunikation, Zuordnen von Objektreferenzen zu Objektkomponentierungen und das Registrieren von Objektkomponentierungen.

Das ORB Interface ermöglicht die direkte Kommunikation mit dem ORB, ist für alle ORBs gleich und unabhängig von den Schnittstellen der Objekte. Hierüber werden einige

Funktionalitäten des ORB angeboten, die sowohl für Clients als auch für Objektimplementierungen nützlich sind.

Interoperabilität

Unter dem Begriff Interoperabilität ist die Verbindung zwischen Objekten verschiedener Plattformen zu verstehen. Das abstrakte General Inter-ORB Protocol (GIOP) ist die Basis der Interoperabilität und definiert allgemeine Protokollspezifikationen, wie z. B. die Darstellung von Datentypen. Ausgehend davon wurden Implementierungsregeln für TCP/IP entwickelt, die im Internet Inter-ORB Protocol (IIOP) definiert sind.

GIOP bildet also die einheitliche Basis für jede CORBA-Kommunikation (auch für Netzprotokolle die nicht auf TCP/IP aufbauen) und IIOP ist eine Verfeinerung für die auf TCP/IP basierende Kommunikation.

Werden in CORBA Objekte z. B. als Parameter oder Rückgabewert verwendet, so werden sie nicht etwa als Original oder Kopie übergeben, sondern als Objektreferenz (= Verweis auf ein Objekt). Da diese Objektreferenz über alle Plattformen und weltweit eindeutig sein muss, heißt sie Interoperable Object Reference (IOR). Die IOR ist aus folgenden Elementen aufgebaut:

- IP-Adresse des Rechners, auf dem der ORB installiert ist
- Portnummer des ORB
- ORB-interne Identifikation

Die Spezifikation der Interoperable Object Reference (IOR) ist von der OMG in zwei Teile gegliedert worden:

- Allgemeiner Teil, der unabhängig von der Art der Netzverbindung gültig und im GIOP definiert ist.
- Netzspezifischer Teil, der definiert wird durch das IIOP.

CORBAServices

Die CORBAServices sind eine Bibliothek von Standard-Diensten. Sie erleichtern und vereinheitlichen die Entwicklung komplexer Anwendungen. Jeder dieser Dienste ist in IDL beschrieben und damit als CORBA-Dienst mit einheitlicher Schnittstelle einsetzbar. Beispiele für CORBAServices sind:

- Naming-Service
- Event-Service
- Life-Cycle-Service

Es gibt in der CORBA-Spezifikation weit mehr CORBAServices als die genannten, diese sind aber bis jetzt nicht implementiert worden.

2.5 Enterprise JavaBeans

Nach Sun Microsystems spezifiziert Enterprise JavaBeans (EJB) eine Architektur für komponentenbasierte, verteilte Java-Anwendungen. EJB ist dabei kein Produkt, sondern lediglich eine Spezifikation, die vorgibt wie Entwickler EJB-Anwendungen zu implementieren haben.

2.5.1 Überblick

Der Begriff Enterprise deutet daraufhin, dass sich EJB im Bereich unternehmensweiter Anwendungen positioniert. Diese Anwendungen werden in Java implementiert. Beans stehen für Komponenten.

Analog zu JavaBeans im Bereich der Benutzeroberflächen-Komponenten, werden EJB im Bereich der Server-Komponenten eingesetzt. Die Enterprise JavaBeans beinhalten die Anwendungslogik und existieren server-seitig auf einem Application-Server. Dabei können Anwendungen aus mehreren Beans bestehen.

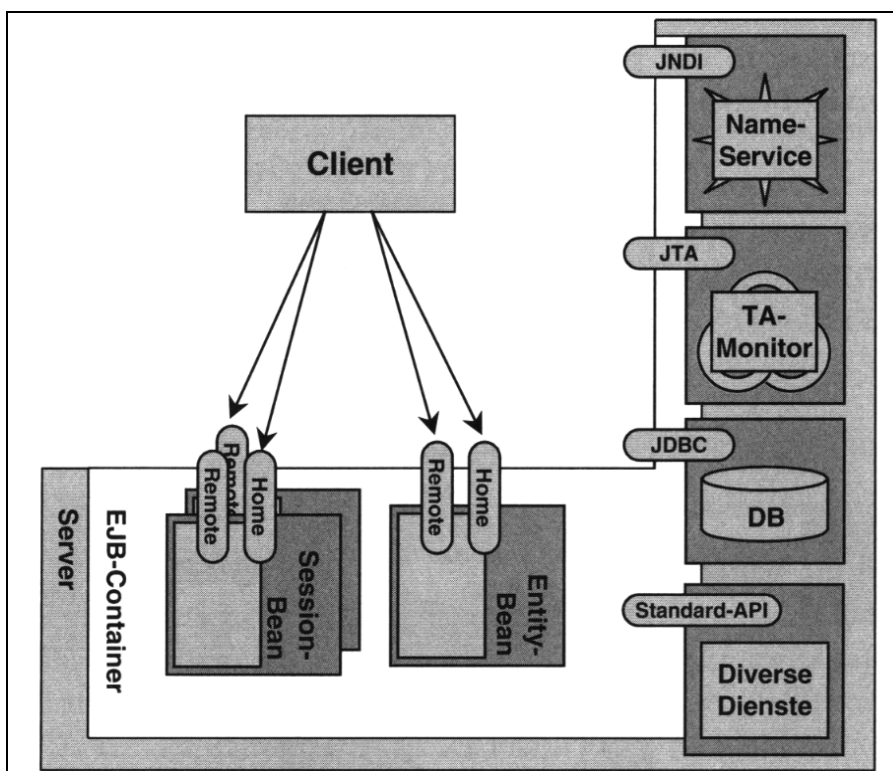
Durch die komponentenbasierte Architektur wird es dem Entwickler ermöglicht, auf zahlreiche Komponenten zurückzugreifen, die von anderen Entwicklern bereits nach der EJB-Spezifikation entwickelt wurden. EJBs können also wiederverwendet werden und ermöglichen dadurch eine sehr effektive Anwendungsentwicklung.

Die EJB-Spezifikation ist Bestandteil der Java2-Plattform Enterprise Edition (J2EE). Die momentan aktuelle Version der EJB-Spezifikation ist die Version 1.1. Der OAS 4.0.8.1 unterstützt allerdings nur die EJB-Spezifikation Version 1.0.

2.5.2 Architektur

In Abbildung 13 wird die Architektur gezeigt, auf der EJB-Anwendungen aufgebaut werden.

Abbildung 13: EJB-Architektur



Quelle: Denninger, Stefan und Ingo Peters: Enterprise JavaBeans. 1. Aufl. München 2000, S. 21.

In einem EJB-Server werden ein oder mehrere EJB-Container ausgeführt. Ein EJB-Container stellt einer Enterprise JavaBean eine Laufzeitumgebung zur Verfügung.

Im Folgenden werden die Bestandteile einer Enterprise JavaBean und die Dienste, die der EJB-Server den Beans zur Verfügung stellt, erläutert.

Bestandteile einer EJB

Eine Enterprise JavaBean besteht laut EJB-Spezifikation immer aus den folgenden Bestandteilen:

- (1) Home-Interface
- (2) Remote-Interface
- (3) Bean-Klasse
- (4) Deployment-Descriptor

(1) Home-Interface

Im Home-Interface einer Enterprise JavaBean definiert der Bean-Provider (derjenige, der die Bean entwickelt) die Signaturen der Methoden, die den Lebenszyklus einer Bean betreffen. Das Home-Interface muss vom Interface "javax.ejb.EJBHome" abgeleitet werden. Ein Beispiel für ein Home-Interface befindet sich in Kapitel "3.7.5 EJB-Anwendungen".

Der EJB-Container implementiert automatisch das Home-Interface beim Installieren der Bean. Hierbei werden Methoden zum Finden¹, Erzeugen und Löschen von Bean-Instanzen generiert. Diese generierte Klasse wird als EJBHome-Klasse bezeichnet. Eine Instanz (ein Objekt) einer EJBHome-Klasse wird als EJBHome bezeichnet. Der Client benutzt das EJBHome um Beans zu finden und um Bean-Instanzen zu erzeugen.

¹ Nur bei Entity-Beans. Siehe Abschnitt (3) Bean-Klasse.

(2) Remote-Interface

Im Remote-Interface einer Enterprise JavaBean definiert der Bean-Provider die Signaturen der Methoden, die von einer Bean nach außen hin angeboten werden. Das Remote-Interface muss vom Interface "javax.ejb.EJBObject" abgeleitet werden. Ein Beispiel für ein Remote-Interface befindet sich in Kapitel "3.7.5 EJB-Anwendungen".

Der EJB-Container implementiert automatisch das Remote-Interface beim Installieren der Bean. Hierbei werden nicht die Methoden der Bean generiert, sondern Code zum Delegieren der Methodenaufrufe an die Bean. Diese generierte Klasse wird als EJBObject-Klasse bezeichnet. Eine Instanz (ein Objekt) einer EJBObject-Klasse wird als EJBObject bezeichnet. Der Client benutzt das EJBObject um die Methoden einer Bean aufzurufen.

(3) Bean-Klasse

In der Bean-Klasse implementiert der Bean-Provider die Methoden, deren Signaturen im Remote-Interface definiert wurden, sowie zusätzliche Methoden, die nicht nach außen hin angeboten werden. Wie in Abbildung 13 gezeigt, werden zwei Arten von Beans unterschieden:

- Session-Bean
- Entity-Bean

Tabelle 1 stellt die wesentlichen Unterschiede zwischen einer Session-Bean und einer Entity-Bean dar.

Tabelle 1: Unterschiede zwischen Session-Bean und Entity-Bean

Merkmal	Session-Bean	Entity-Bean
Aufgabe	Repräsentiert einen serverseitigen Dienst, der Aufgaben für einen Client ausführt.	Repräsentiert ein Geschäftsobjekt, dessen Daten sich in einem dauerhaften Speicher befinden.

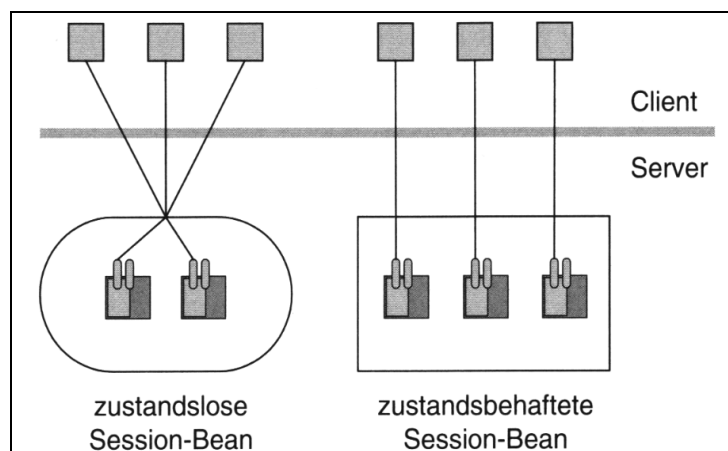
Merkmal	Session-Bean	Entity-Bean
Zugriff	Eine Session-Bean ist für den Client eine private Ressource. Sie steht ihm exklusiv zur Verfügung.	Die Entity-Bean ist eine zentrale Ressource; die Bean-Instanz wird von mehreren Clients gleichzeitig benutzt, und ihre Daten stehen allen Clients zur Verfügung.
Persistenz	Nicht persistent, sondern transient; wenn der verbundene Client oder der Container terminiert werden, ist die Bean nicht mehr verfügbar.	Persistent; wenn die verbundenen Clients oder der Server terminiert werden, befindet sich der Zustand der Entity-Bean auf einem persistenten Speichermedium; die Bean kann so zu einem späteren Zeitpunkt wiederhergestellt werden. Es wird unterschieden zwischen: Bean-managed: Bean macht Daten persistent Container-managed: Container macht die Daten der Bean persistent

Quelle: Denninger, Stefan und Ingo Peters: Enterprise JavaBeans. 1. Aufl. München 2000, S. 31.

Wie in Abbildung 14 dargestellt, lässt sich bei einer Session-Bean noch eine weitere Unterscheidung treffen:

- (a) Stateless (zustandslose) Session-Bean
- (b) Stateful (zustandsbehaftete) Session-Bean

Abbildung 14: Stateless und stateful Session-Bean



Quelle: Denninger, Stefan und Ingo Peters: Enterprise JavaBeans. 1. Aufl. München 2000, S. 68.

(a) Stateless (zustandslose) Session-Bean

Der EJB-Container verwaltet einen Pool von stateless Session-Beans. Er ordnet für jeden Methodenaufruf dem entsprechenden Client eine existierende oder eine neue stateless Session-Bean aus dem Pool zu. Eine stateless Session-Bean speichert von einem Methodenaufruf zum nächsten keine Daten. Die Methoden einer stateless Session-Bean arbeiten nur mit den Daten, die ihr als Parameter übergeben werden. Stateless Session-Beans des gleichen Typs besitzen alle die gleiche Identität. Da sie keinen Zustand haben, besteht weder die Notwendigkeit noch die Möglichkeit, sie zu unterscheiden.

(b) Stateful (zustandsbehaftete) Session-Bean

Eine stateful Session-Bean ist von ihrer Erzeugung bis zur Löschung an einen Client gebunden. Sie speichert Daten über mehrere Methodenaufrufe hinweg. Methodenaufrufe an eine stateful Session-Bean können den Zustand der Bean verändern. Der Zustand geht verloren wenn der Client die Bean nicht mehr benutzt, oder der Server heruntergefahren wird. Stateful Session-Beans des gleichen Typs haben unterschiedliche Identitäten. Der EJB-Container muss sie unterscheiden können, da sie für ihre Clients jeweils unterschiedliche Zustände verwalten.

Zu beachten ist, dass die aktuelle Version 4.0.8.1 des OAS lediglich den Einsatz von Session-Beans unterstützt. Ein Beispiel für eine Session-Bean-Klasse befindet sich in Kapitel "3.7.5 EJB-Anwendungen".

(4) Deployment-Descriptor

Zu jeder Bean muss der Bean-Provider einen Deployment-Descriptor erstellen. Der Deployment-Descriptor enthält Informationen über die Struktur einer Enterprise JavaBean und ihrer externen Abhängigkeiten. Zusätzlich enthält er Informationen darüber, wie der EJB-Container die Bean zur Laufzeit zu behandeln hat und wie die Bean mit anderen Beans zu komplexen Komponenten kombiniert werden kann.

Ein Beispiel für einen Deployment-Descriptor befindet sich in Kapitel "3.7.5 EJB-Anwendungen".

Alle Bestandteile einer Enterprise JavaBean müssen vom Bean-Provider in ein ejb-jar-File gepackt werden. Dieses ejb-jar-File wird entweder an den Bean-Deployer (derjenige, der die Bean installiert) oder den Application-Assembler (derjenige, der die Bean in eine komplexe Anwendung einbaut) übergeben.

Dienste

Wie Abbildung 13 zeigt, stellt der EJB-Server dem (den) Container(n) folgende Dienste über Standard-Schnittstellen zur Verfügung:

- (1) Naming-Service
- (2) TA-Monitor
- (3) Datenbank

(1) Naming-Service

Der Naming-Service ist ein Namensdienst, mit dessen Hilfe Beans aufgefunden werden können. Er verwaltet Name-Wert-Paare in einem Namensraum. Das EJB-Konzept verwendet den Namensdienst zur Verwaltung der Home-Interfaces. In jedem der Name-Wert-Paare ist der Bean-Name als Name und die EJBHome-Referenz als Wert gespeichert. Ein Client kann sich vom Namensdienst die EJBHome-Referenz einer Bean geben lassen, wenn er den Namen der Bean kennt.

Der Naming-Service wird über das JNDI (Java Naming and Directory Interface)¹ angesprochen.

(2) TA-Monitor

In einer verteilten objektorientierten Anwendung fallen, ähnlich wie bei einem DBMS, Aufgaben an, wie das Sicherstellen der Datenkonsistenz, Commit und Rollback. Der TA-Monitor stellt den Beans eine Umgebung zur Verfügung, die den Umgang mit Transaktionen unterstützt.

¹ Siehe Glossar im Anhang.

Die EJB-Spezifikation unterstützt verteilte Transaktionen. Dabei kann eine Transaktion sowohl unterschiedliche Datenbanken, als auch unterschiedliche EJB-Server betreffen. Der TA-Monitor wird über das JTA (Java Transaction API)¹ angesprochen.

(3) Datenbank

Bean-Instanzen können über JDBC (Java Database Connection)² auf verschiedene Datenbanken zugreifen. Voraussetzung ist, dass der Hersteller der Datenbank einen JDBC-Treiber liefert.

¹ Siehe Glossar im Anhang.

² Siehe Glossar im Anhang.

3 Oracle Application-Server

In diesem Kapitel wird zunächst beschrieben was unter einem Application-Server allgemein zu verstehen ist und welche Middleware-Funktionen ein Application-Server bietet. Danach wird speziell auf den Oracle Application-Server (OAS) in der Version 4.0.8.1 eingegangen. Es werden Themen wie Skalierbarkeit, Lastverteilung, OAS-Manager, Datenbankzugriff, Interaktionsmodelle und Sicherheit betrachtet. Im Anschluß daran werden die Einsatzmöglichkeiten des OAS ausführlich dargestellt, d. h. welche Anwendungstypen eingesetzt werden können.

3.1 Einführung

Nach der in der IT-Welt akzeptierten Definition der Gartner Group¹ wird der Begriff Application-Server ganz abstrakt definiert:

Def.: Application-Server = Eine Umgebung in der Anwendungslogik ausgeführt wird.

Der Application-Server ist auf der Mittel-Schicht einer 3-Schicht-Client/Server-Architektur implementiert. Die Mittel-Schicht fungiert hierbei als Schnittstelle zwischen Client und DBMS.

Aufgabe des Application-Server ist die zentrale Haltung und Ausführung von Anwendungen. Somit hat er die Rolle eines zentralen Anwendungscontainers. Das gesamte Anwendungsmanagement, vor allem was die Verteilung von neuer Software oder Software-Updates angeht, geschieht zentral und somit kostengünstig auf dem Application-Server. Die durchgeführten Änderungen stehen sofort allen Benutzern der Anwendungen zur Verfügung.

Auf dem Client werden ausschließlich die Ergebnisse der ausgeführten Anwendungslogik präsentiert. Die Präsentation ist also von der Ausführung entkoppelt. Als Client genügt ein

¹ Siehe Glossar im Anhang.

Web-Browser, der Anfragen an den Application-Server sendet und die erhaltenen Antworten anzeigt.

Von der Anwendungslogik zur Verarbeitung benötigte Daten werden von DBMS-Servern auf der 1. Schicht geliefert. Somit ist ein direkter Zugriff auf Datenquellen vom Client nicht mehr erforderlich.

Anwendungen im WWW oder Intranet erfordern heute weitergehende Anforderungen wie Skalierbarkeit, Zuverlässigkeit, Sicherheit und Transaktionsdienste, die durch Middleware-Funktionen erfüllt werden. Diese Middleware-Funktionen wurden in der Vergangenheit durch die Kombination von mehreren Middleware-Produkten abgedeckt, die jeweils auf eine Funktion ausgerichtet sind. Ein Application-Server vereinigt diese Middleware-Produkte in einem Produkt. Im Folgenden wird auf die Middleware-Funktionen eingegangen, die ein Application-Server unterstützt.

Meistens ist die konkrete Anzahl der Benutzer einer Anwendung nicht vorhersehbar, besonders wenn sie über das WWW genutzt wird. Ein Application-Server stellt Mechanismen zur Skalierbarkeit bereit, um flexibel auf die aktuelle Belastungssituation reagieren zu können und um die vorhandenen Ressourcen optimal zu nutzen.

Weiterhin kann mit einem Application-Server die Zuverlässigkeit der eingesetzten Anwendungen gewährleistet werden, indem dieser im Fehlerfall die Verfügbarkeit einer Anwendung möglichst schnell wiederherstellt. Denn ein dauerhafter Ausfall einer Anwendung im WWW bedeutet meistens hohe Verluste für ein Unternehmen und sinkende Akzeptanz der Anwendung bei den Benutzern bzw. Kunden.

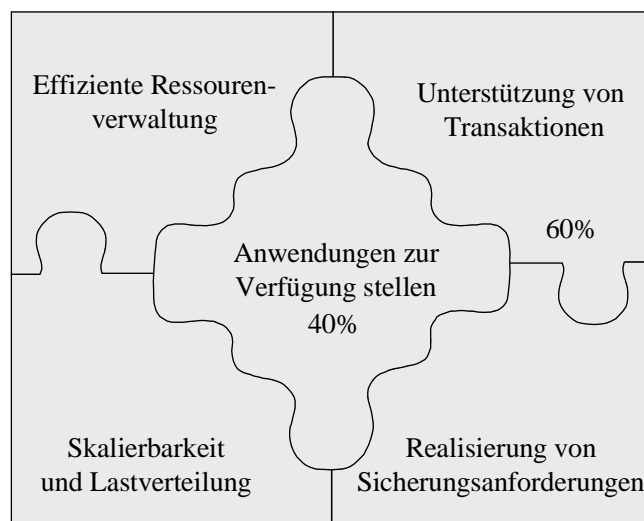
Ferner spielt die Performance von Anwendungen eine entscheidende Rolle. Viele Benutzer bzw. Kunden sehen in schlechten Antwortzeiten einen schlechten Service. Durch dynamische Lastverteilung der eingehenden Anfragen und einer parallelen Durchführung unterschiedlicher Arbeitsschritte werden mit einem Application-Server hohe Ausführungsgeschwindigkeiten erreicht.

Der Application-Server kann auch die Sicherheit der Datenübertragung zwischen den einzelnen Schichten gewährleisten. Dazu bietet er die Möglichkeit zu übertragende Daten nach bestimmten Verfahren wie z. B. SSL (Secure Sockets Layer) zu verschlüsseln.

Zusätzlich stellt der Application-Server Möglichkeiten zur Authentifizierung von Benutzern bereit und kontrolliert somit den Zugriff auf Ressourcen und Datenquellen.

Die Bereitstellung und Abwicklung dieser Middleware-Funktionen ist die Hauptaufgabe von Application-Servern und macht ca. 60 % des Gesamteinsatzes aus. In Abbildung 15 wird gezeigt, wie die Aufgaben eines Application-Server verteilt sind.

Abbildung 15: Aufgaben eines Application-Server



Quelle: In Anlehnung an: Geiger, Erwin: Leistungsfähige Mittler im Netz. In: IT-Journal Ausgabe 06/99, S. 70.

Der größte Vorteil beim Einsatz eines Application-Server ist, dass die oben beschriebenen Middleware-Funktionen nicht explizit in den Anwendungen von den Entwicklern implementiert werden müssen, sondern als Dienste vom Application-Server bereitgestellt werden.

Eine weitere, immer bedeutender werdende Middleware-Funktion eines Application-Server ist die Unterstützung der Komponentenmodelle CORBA und EJB.

Oracle Application-Server

Mit dem Oracle Application-Server in der Version 4.0.8.1 stellt Oracle dem Benutzer eine Plattform für den Einsatz server-seitiger Anwendungen für das WWW, Intranet und

Extranet zur Verfügung. Diese Anwendungen können dabei in verschiedenen Programmiersprachen wie Java, Perl, C oder PL/SQL geschrieben sein.

Der OAS wird in zwei unterschiedlichen Versionen von Oracle vertrieben. Neben der Standard Edition mit eingeschränkter Funktionalität wird auch eine doppelt so teure Enterprise Edition angeboten. Die Enterprise Edition bietet zusätzlich zur Standard Edition die Anbindung von ODBC-Datenquellen und die Unterstützung von Transaktionen.

Der Bericht "1999 Worldwide Market for Application-Servers: Setting the Course for 2000" der International Data Corporation (IDC) weist Oracle als führenden Anbieter von Application-Servern aus.

3.2 Architektur

Bei der Installation des OAS sind zwei Konfigurationen möglich. Die erste Möglichkeit ist die "Single-Node-Konfiguration". Hierbei werden alle Komponenten des OAS auf einem Rechner (= Single-Node) installiert.

Die zweite Möglichkeit ist die Multi-Node-Konfiguration. Hierbei werden die Komponenten des OAS verteilt auf einem Verbund mehrerer Rechner (= Multi-Node) installiert. Die Multi-Node-Konfiguration besteht aus einem Primary-Node, der auf genau einem Rechner installiert wird, und in der Regel aus mehreren Remote-Nodes, die auf mehreren Rechnern installiert werden.

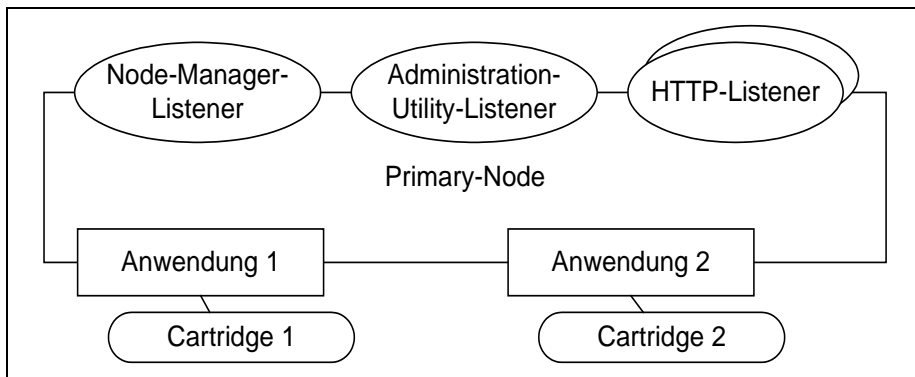
Eine Installation eines OAS wird als OAS-Web-Site bezeichnet. Jede OAS-Web-Site hat einen Web Request Broker (WRB). Der WRB ist ein OAS-Runtime-Prozess (kurz: OAS-Runtime), der die Ressourcen des OAS verwaltet.

In Abbildung 16 werden die Komponenten eines Primary-Node einer OAS-Web-Site dargestellt. Ein Primary-Node einer OAS-Web-Site hat immer einen Node-Manager-Listener, einen Administration-Utility-Listener und beliebig viele HTTP-Listener¹.

¹ Siehe Kapitel 3.2.1 HTTP-Listener-Ebene.

Weiterhin können auf einem Primary-Node beliebig viele Anwendungen¹ installiert sein, die ihrerseits Cartridges² enthalten.

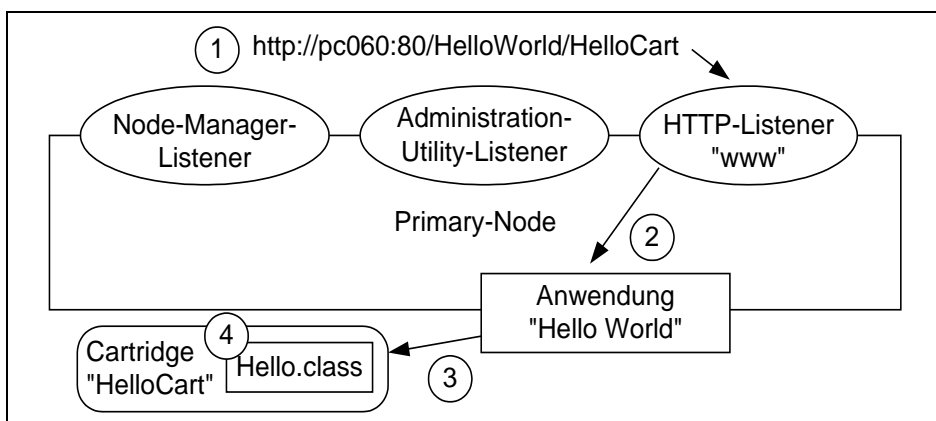
Abbildung 16: Komponenten des Primary-Node einer OAS-Web-Site



Quelle: Oracle Application Server Documentation, Release 4.0.8.1: Administration Guide, S. 1-2.

Im Folgenden soll die Funktionsweise des OAS näher betrachtet werden. Zur Verdeutlichung der Funktionsweise zeigt Abbildung 17 die Schritte, die zur Bearbeitung einer Anfrage an eine Anwendung durchgeführt werden.

Abbildung 17: Bearbeitung einer Anfrage an eine Anwendung durch den OAS



Quelle: Oracle Application Server Documentation, Release 4.0.8.1: Administration Guide, S. 1-3.

¹ Siehe Kapitel 3.2.3.2 Anwendung.

² Siehe Kapitel 3.2.3.1 Cartridge.

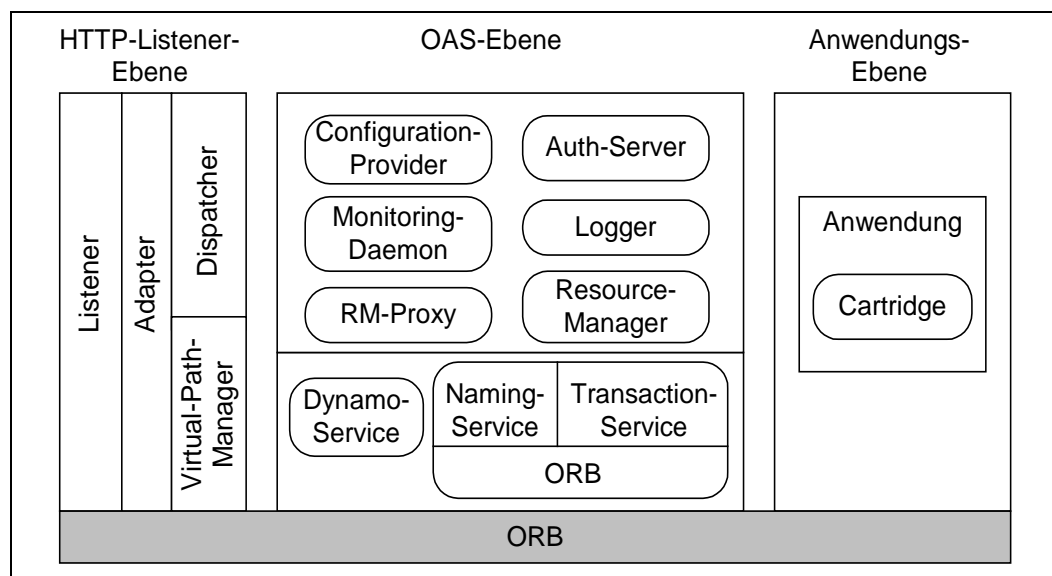
Bearbeitung der Anfrage:

1. Der Listener "www", der auf Port 80 lauscht, empfängt die Anfrage "http://pc060:80/HelloWorld/HelloCart" des Client.
2. Der OAS erkennt anhand der Anfrage, dass die Anwendung "HelloWorld" angesprochen werden soll. Diese erzeugt eine Runtime-Umgebung für die Cartridge "HelloCart". Da in diesem Beispiel die Cartridge "HelloWorld" eine JServlet-Cartridge ist, wird eine Java Virtual Machine gestartet.
3. Die Anwendung "HelloWorld" startet die Cartridge "HelloCart".
4. Die Cartridge "HelloCart" führt die Anwendungslogik der Klasse "Hello.class" aus.

Im Folgenden soll die Architektur des OAS näher beschrieben werden. Die Architektur des OAS kann durch ein Modell beschrieben werden, dass in 3 Ebenen (Layer) unterteilt ist:

- HTTP-Listener-Ebene
- OAS-Ebene
- Anwendungsebene

Abbildung 18: OAS-Architektur



Quelle: Oracle Application Server Documentation, Release 4.0.8.1: Overview and Glossary, S. 2-2.

In Abbildung 18 werden diese Ebenen veranschaulicht. Die Ebenen kommunizieren untereinander über den Object Request Broker (ORB). Jede Ebene besteht ihrerseits aus einzelnen Komponenten, die in den nachfolgenden Kapiteln beschrieben werden.

3.2.1 HTTP-Listener-Ebene

Auf der HTTP-Listener-Ebene gehen alle Anfragen der Clients über das HTTP ein. Sie besteht aus den folgenden Komponenten:

- (1) Listener
- (2) Adapter
- (3) Dispatcher
- (4) Virtual-Path-Manager

(1) Listener

Ein Listener ist ein Web-Server und somit der Einstiegspunkt in den OAS. Er empfängt die Anfragen (Requests) und leitet sie an den Adapter weiter. Tabelle 2 zeigt welche Listener auf einem Primary-Node immer installiert werden.

Tabelle 2: Listener und Default Ports eines Primary-Node

Listener	Default Port
Node-Manager-Listener (Default-Name: "nodemgr")	8888
Administration-Utility-Listener (Default-Name: "admin")	8889
HTTP-Listener (Default-Name: "www")	80

Der "Node-Manager-Listener" ist auf jedem Node einer OAS-Web-Site präsent. Über den Node-Manager-Listener wird die "OAS Welcome Page" aufgerufen. Von dort können der OAS-Manager und die OAS-Utilities aufgerufen werden.

Der OAS-Manager¹ ist ein graphisches Werkzeug zur Administration und Konfiguration der OAS-Web-Site. Mit Hilfe der OAS-Utilities wird das "PL/SQL-WebToolkit"², der "Log-Analyzer" sowie der "Database Browser" installiert. Mit dem Log-Analyzer können Log-Informationen analysiert werden, die der OAS erstellt hat. Mit dem Database Browser können durch Angabe eines DAD³ die Objekte einer Datenbank betrachtet werden.

Über den Listener "Administration-Utility-Listener" können der Log-Analyzer oder der Database Browser ausgeführt werden. Er wird über den OAS-Manager administriert und konfiguriert.

Über den "HTTP-Listener" werden die Anfragen der Clients an die installierten Anwendungen empfangen. Er wird über den OAS-Manager administriert und konfiguriert.

(2) Adapter

Der Adapter ist ein API (Application Programming Interface), das vom HTTP-Listener benutzt wird, um sich mit dem Dispatcher zu verbinden.

(3) Dispatcher

Der Dispatcher ist ein Verteiler. Es gibt zu jedem Listener immer genau einen Dispatcher. Empfängt ein HTTP-Listener eine Anfrage, die sich an eine Cartridge richtet, wird die Anfrage an den Dispatcher übergeben. Dieser kontaktiert den Virtual-Path-Manager. Aufgrund der Information vom Virtual-Path-Manager wird die Anfrage direkt an den richtigen Cartridge-Typ weitergegeben, oder zuvor der Auth-Server zur Authentifizierung kontaktiert.

¹ Siehe Kapitel "3.3 OAS-Manager".

² Siehe Kapitel "3.7.1 PL/SQL-Anwendungen".

³ Siehe Kapitel "3.4 Datenbankzugriff".

(4) Virtual-Path-Manager

Der Virtual-Path-Manager wählt aufgrund der Anfrage den geeigneten Cartridge-Typ aus und gibt diese Information zurück an den Dispatcher. Betrifft die Anfrage einen geschützten virtuellen Pfad, teilt der Virtual-Path-Manager dem Dispatcher in Form eines Authentication-String¹ mit, welches Authentifizierungsschema zur Authentifizierung benutzt werden soll.

3.2.2 OAS-Ebene

Die OAS-Ebene ist die komplexeste Ebene. Die Komponenten der OAS-Ebene sind in zwei Gruppen eingeteilt:

- (1) OAS-Komponenten
- (2) ORB-Komponenten

(1) OAS-Komponenten

Die OAS-Komponenten stellen ein Ressourcen-Management dar. Das heißt auf der OAS-Ebene werden die Ressourcen für Anfragen an Anwendungen zugeteilt. Die OAS-Komponenten sind Prozesse. Tabelle 3 zeigt die OAS-Komponenten.

Tabelle 3: OAS-Komponenten

Komponente	Beschreibung
Authentication-Server (kurz: Auth-Server) (Prozess: "wrbasrv")	Kapselt die Authentifizierungsschemata. Ein Authentication-Server besteht aus einem Authentication-Broker (kurz: Auth-Broker) und mehreren Authentication-Providern (kurz: Auth-Provider) ² .
Configuration-Provider (Prozess: "wrbcfg")	Liest Konfigurationsinformationen aus der Datei "wrb.app" und liefert sie an die Komponente des OAS, die sie benötigt.

¹ Siehe Kapitel "3.6.1 Authentifizierung".

² Siehe Kapitel "3.6.1 Authentifizierung".

Komponente	Beschreibung
Logger (Prozess: "wrblog")	Ermöglicht den Anwendungen Fehler und Warnmeldungen in ein zentrales Repository zu schreiben.
Monitoring-Daemon (Prozess: "wrbmon")	Überwacht die Zustände der OAS-Prozesse.
Ressource-Manager (kurz: RM) (Prozess: "wrbroker")	Verwaltet die OAS-Prozesse und wacht über die richtige Anzahl von Cartridge-Servern und Cartridge-Instanzen ¹ . Pro OAS-Web-Site gibt es einen Ressource-Manager.
RM-Proxy (Prozess: "wrbrmpxy")	Dienst des WRB. Erhält die Referenzen auf EJB- und C++-Objekte und gibt diese zurück an die Clients ² .

Quelle: In Anlehnung an: Oracle Application Server Documentation, Release 4.0.8.1: Overview and Glossary, S. 2-4.

(2) ORB-Komponenten

Der Object Request Broker³ (ORB) des OAS verwaltet die Kommunikation zwischen den Ebenen. Die ORB-Komponenten sind Prozesse. Tabelle 4 listet die ORB-Komponenten auf und beschreibt sie.

Tabelle 4: ORB-Komponenten

Komponente	Beschreibung
ORB (Prozess: "oasoorb")	Der ORB-Runtime-Prozess. (kurz: ORB-Runtime)
Naming-Service (Prozess: "oasoorb")	Der Naming-Service ⁴ ist Teil der ORB-Runtime. Er gibt eine Referenz auf das angefragte Objekt zurück.
Transaction-Service (Prozess: "oasoorb")	Der Transaction-Service ⁵ ist Teil der ORB-Runtime. Er verwaltet eine Menge von DB-Operationen als eine Einheit.

¹ Siehe Kapitel "3.2.3.1 Cartridge".

² Siehe Kapitel "3.7.5 EJB-Anwendungen" bzw. "3.7.4 C++-CORBA-Anwendungen".

³ Siehe Glossar im Anhang.

⁴ Siehe Kapitel "3.7.5 EJB-Anwendungen".

⁵ Siehe Kapitel "3.5.3 Transaktion".

Komponente	Beschreibung
Dynamo-Service (Prozess: "oasdyn")	Der Prozess der die Performance des ORB überwacht.

Quelle: In Anlehnung an: Oracle Application Server Documentation, Release 4.0.8.1: Overview and Glossary, S. 2-4.

3.2.3 Anwendungsebene

Auf der Anwendungsebene werden die Anwendungen installiert, die auf dem OAS eingesetzt werden sollen. Die Anwendungsebene besteht aus den zwei Komponenten:

- Cartridge
- Anwendung

Diese Komponenten werden in den beiden nachfolgenden Kapiteln erklärt und ihre Zusammenhänge dargestellt.

3.2.3.1 Cartridge

Ein Cartridge besteht zum einen aus Code, der Anwendungslogik enthält, und zum anderen aus Parametern, mit denen die Cartridge für die Ausführung konfiguriert wird.

Einige bestimmte Cartridges bieten zusätzlich eine Laufzeit-Umgebung (Runtime Enviroment). So enthält zum Beispiel eine JServlet-Cartridge eine Java Virtual Machine (JVM) zum Ausführen von Java-Klassen und die Perl-Cartridge einen Perl-Interpreter zum Ausführen von Perl-Skripten. Die Cartridges sind innerhalb einer Anwendung enthalten.

Der OAS 4.0.8.1 bietet folgende Cartridge-Typen, mit deren Hilfe entsprechende Anwendungen auf dem OAS eingesetzt werden können:

- PL/SQL

- JServlet
- LiveHTML
- Perl
- C

Zum Ausführen einer Cartridge wird eine Cartridge-Instanz erzeugt und diese in einem Cartridge-Server ausgeführt. Im folgenden Kapitel werden diese Vorgänge ausführlich beschrieben.

Cartridge-Server und Cartridge-Instanz

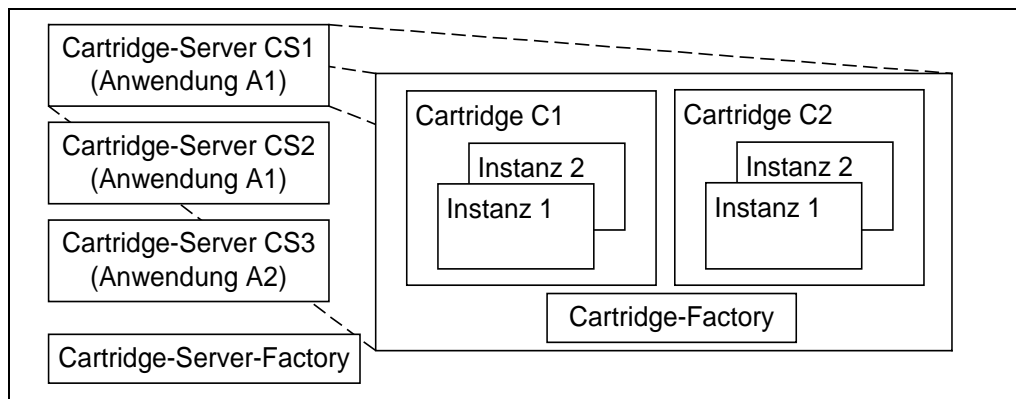
Ein Cartridge-Server ist das Laufzeit-Äquivalent einer Anwendung und kann daher mit einer Anwendung gleichgesetzt werden. Jeder Cartridge-Server kann demnach nur eine Anwendung ausführen. Cartridge-Server werden durch eine "Cartridge-Server-Factory" (Prozess: "wrksf") erzeugt. Ein Cartridge-Server wird als Prozess ausgeführt, innerhalb dessen eine oder mehrere Cartridge-Instanzen erzeugt und danach ausgeführt werden können.

Ein Cartridge-Server erzeugt mit Hilfe seiner "Cartridge-Factory" (Prozess: "wrbfac") Cartridge-Instanzen. Eine Cartridge-Factory kann somit als Konstruktor für Cartridge-Instanzen angesehen werden. Jeder Cartridge-Server hat genau eine Cartridge-Factory. Die erzeugten Cartridge-Instanzen bearbeiten die Anfragen der Clients.

Mehrere Cartridge-Server und mehrere Cartridge-Instanzen werden zum Zwecke der Lastverteilung genutzt. Erhöht sich die Anzahl der Clients, die auf eine Anwendung zugreifen, erhöht der OAS auch die Anzahl der Cartridge-Server bzw. Cartridge-Instanzen.

In Abbildung 19 wird anhand eines Beispiels gezeigt, wie die Begriffe Anwendung, Cartridge, Cartridge-Server, Cartridge-Server-Factory, Cartridge-Instanz und Cartridge-Factory zusammenhängen.

Abbildung 19: Cartridge-Server und Cartridge-Instanzen



Quelle: Oracle Application Server Documentation, Release 4.0.8.1: Overview and Glossary, S. 2-6.

Die drei Cartridge-Server "CS1", "CS2" und "CS3" in Abbildung 19 werden auf einem Node ausgeführt. "CS1" und "CS2" werden für die Anwendung "A1" und "CS3" wird für die Anwendung "A2" ausgeführt. Im Cartridge-Server "CS1" werden die zwei Cartridges "C1" und "C2" mit jeweils zwei Cartridge-Instanzen ausgeführt. Instanziiert werden die Cartridges "C1" und "C2" durch die Cartridge-Factory von Cartridge-Server "CS1".

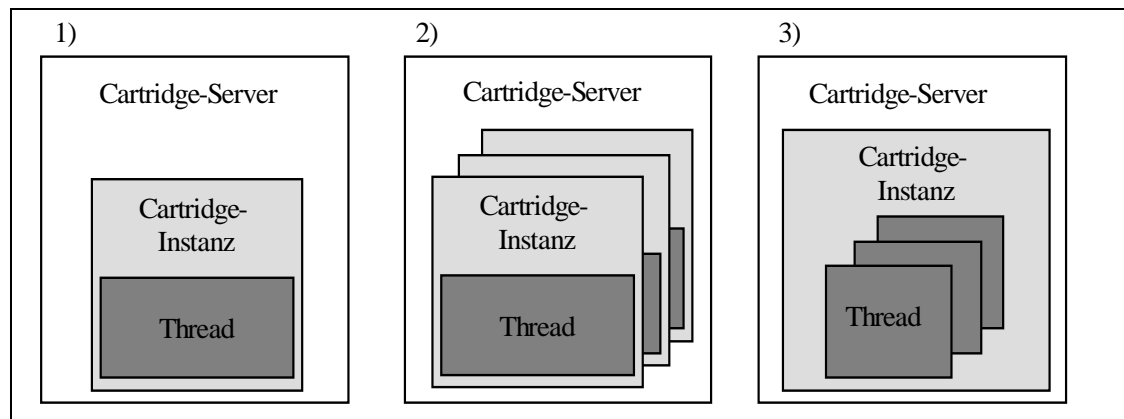
Ein Cartridge-Server ist multithread-safe. Das heißt er kann eine Cartridge-Instanz in mehreren Threads ausführen. Dies ist allerdings nur dann möglich, wenn auch der Code multithread-safe ist, aus dem die Cartridge besteht.

Je nach Cartridge-Typ verwenden die Cartridge-Server verschiedene Thread-Modelle. In Abbildung 20 werden die einzelnen Thread-Modelle dargestellt, die von Cartridge-Servern in Abhängigkeit vom Cartridge-Typ verwendet werden.

Folgende Thread-Modelle werden je nach Cartridge-Typ von einem Cartridge-Server verwendet:

- (1) Ein Thread pro Cartridge-Instanz, eine Cartridge-Instanz pro Cartridge-Server
- (2) Ein Thread pro Cartridge-Instanz, mehrere Cartridge-Instanzen pro Cartridge-Server
- (3) Mehrere Threads pro Cartridge-Instanz, eine Cartridge-Instanz pro Cartridge-Server

Abbildung 20: Thread-Modelle der Cartridge-Server

**(1) Ein Thread pro Cartridge-Instanz, eine Cartridge-Instanz pro Cartridge-Server**

Jeder Cartridge-Server hat genau eine Cartridge-Instanz. Diese Cartridge-Instanz wird in genau einem Thread ausgeführt. Perl-Cartridges und LiveHTML-Cartridges arbeiten nach diesem Thread-Modell.

(2) Ein Thread pro Cartridge-Instanz, mehrere Cartridge-Instanzen pro Cartridge-Server

Jeder Cartridge-Server hat mehrere Cartridge-Instanzen. Jede Cartridge-Instanz wird in genau einem Thread ausgeführt. Nur PL/SQL-Cartridges arbeitet nach diesem Thread-Modell.

(3) Mehrere Threads pro Cartridge-Instanz, eine Cartridge-Instanz pro Cartridge-Server

Jeder Cartridge-Server hat genau eine Cartridge-Instanz. Jede Instanz wird in mehreren Threads ausgeführt. Das hat den Vorteil, dass mehrere Anfragen gleichzeitig bearbeitet werden können, wobei jeder Thread genau eine Anfrage bearbeitet. Der Code aus dem die Cartridge besteht, muss multithread-safe sein. JServlet-Cartridges, C-Cartridges, C++-CORBA-Cartridges, sowie EJB-Objekte arbeiten nach diesem Thread-Modell.

3.2.3.2 Anwendung

Eine Anwendung (Application) ist eine server-seitige Komponente, die auf dem OAS ausgeführt wird. Anwendungen können in verschiedenen Programmiersprachen entwickelt und auf dem OAS eingesetzt werden. Eine Anwendung auf dem OAS kann mit Hilfe des OAS-Manager¹ installiert und konfiguriert werden.

Die Anwendungen werden aufgrund der Übertragungsprotokolle, über die Clients mit dem OAS kommunizieren, in zwei Kategorien eingeteilt:

- Cartridge-basierte Anwendung (HTTP)
- Komponenten-basierte Anwendung (IIOP)

Diese beiden Arten von Anwendungen werden in den nächsten beiden Kapiteln kurz vorgestellt und im Kapitel "3.7 Einsatzmöglichkeiten" ausführlich erklärt.

3.2.3.2.1 Cartridge-basierte Anwendung

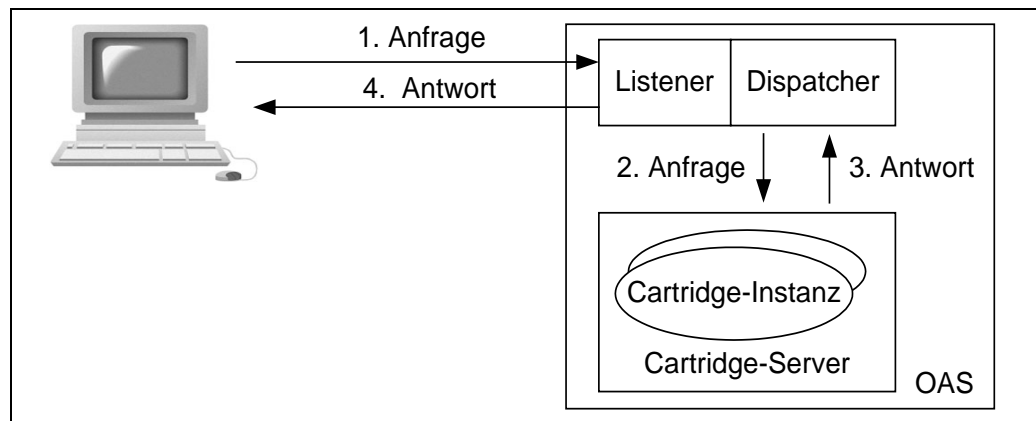
Eine cartridge-basierte Anwendung enthält eine oder mehrere Cartridges des gleichen Typs. Es ist also nicht möglich, dass eine Anwendung beispielsweise eine PL/SQL-Cartridge und eine JServlet-Cartridge enthält. Cartridge-basierte Anwendung verwenden das HTTP.

In Abbildung 21 wird der Weg einer Anfrage von einem Client-Browser an eine cartridge-basierte Anwendung gezeigt.

Die Anfrage wird vom Dispatcher an eine verfügbare Cartridge-Instanz des entsprechenden Typs weitergegeben. Verfügt der Cartridge-Server nicht über eine solche Cartridge-Instanz, erzeugt die Cartridge-Factory eine neue Cartridge-Instanz in dem bestehenden Cartridge-Server, die dann die Anfrage bearbeitet.

¹ Siehe Kapitel "3.3 OAS-Manager".

Abbildung 21: Anfrage an eine cartridge-basierte Anwendung



Hat der Cartridge-Server seine maximale Anzahl an Cartridge-Instanzen erreicht, dann erzeugt die Cartridge-Server-Factory einen neuen Cartridge-Server, in dem dann eine neue Cartridge-Instanz erzeugt wird.

Diese neue Cartridge-Instanz bearbeitet dann die Anfrage. Wenn die Anfrage abgearbeitet ist, steht die Cartridge-Instanz für neue Anfragen bereit.

Eine cartridge-basierte Anwendung wird mit Hilfe des OAS-Manager konfiguriert. Hierbei wird mit Parametern auf zwei Ebenen gearbeitet:

- (1) Anwendungsebene
- (2) Cartridge-Ebene

(1) Anwendungsebene

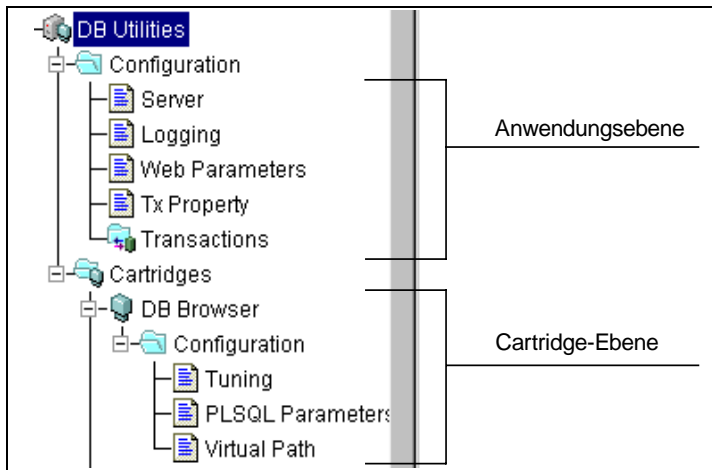
Die Parameter der Anwendungsebene werden auf alle Cartridges angewendet, die zur Anwendung gehören. Die Konfiguration wird somit vereinfacht, da gemeinsame Parameter aller Cartridges nicht für jede Cartridge einzeln gesetzt werden müssen.

(2) Cartridge-Ebene

Auf Cartridge-Ebene werden die Parameter für eine bestimmte Cartridge gesetzt. Die Parameter einer Cartridge werden bei der Ausführung auf alle Cartridge-Instanzen angewendet.

In Abbildung 22 wird dargestellt, wie bei einer PL/SQL-Anwendung mit dem Namen "DB Utilities" diese zwei Ebenen im OAS-Manager angeordnet sind.

Abbildung 22: Parameter einer PL/SQL-Anwendung



Eine ausführliche Beschreibung der cartridge-basierten Anwendungen erfolgt im Kapitel "3.7 Einsatzmöglichkeiten".

3.2.3.2.2 Komponenten-basierte Anwendung

Auf dem OAS können Anwendungen eingesetzt werden, die auf verteilten Objekten beruhen. Diese Objekte basieren auf dem CORBA-Standard und kommunizieren über das IIOP.

In der Version 4.0.8.1 unterstützt der OAS den Einsatz der folgenden komponenten-basierten Anwendungen:

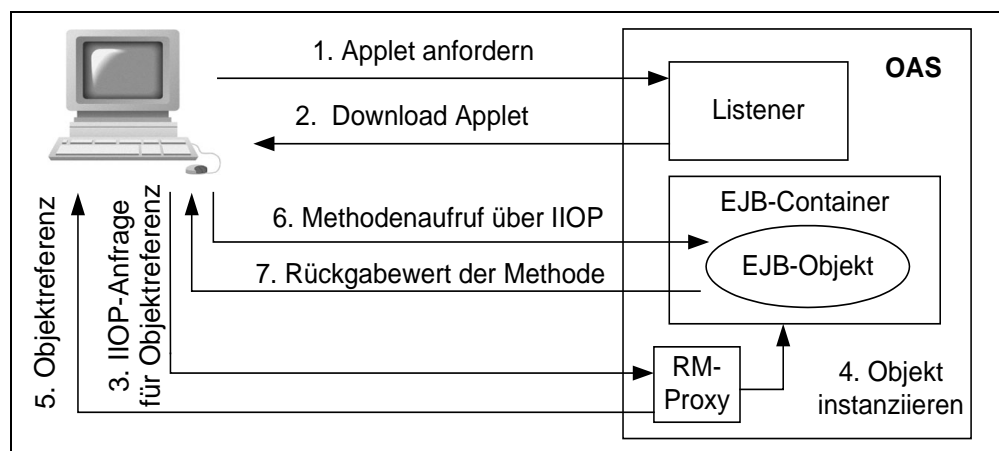
- Enterprise JavaBeans (EJB)
- C++-CORBA

Im Folgenden wird stellvertretend für das Prinzip einer komponenten-basierten Anwendung eine EJB-Anwendung beschrieben.

Bei einer EJB-Anwendung benutzt der OAS keinen Cartridge-Server, sondern einen EJB-Container¹, als Laufzeit-Aquivalent zur EJB-Anwendung, um Instanzen der Anwendung zu erzeugen und auszuführen.

In Abbildung 23 wird gezeigt, wie die Anfrage eines Client (Java-Applet) über das IIOP an ein EJB-Objekt, das auf dem OAS installiert ist, behandelt wird.

Abbildung 23: Methodenaufruf eines EJB-Objektes



Fragt der Client nach dem Herunterladen des Applet eine Objektreferenz an, so beauftragt der RM-Proxy den EJB-Container das betreffende Objekt zu instanzieren (falls nicht bereits eine Instanz existiert). Der RM-Proxy gibt die Objektreferenz an den Client zurück. Der Client benutzt diese Objektreferenz um Methoden des Objektes aufzurufen.

Die Administration und Konfiguration der komponenten-basierten Anwendungen mit Hilfe des OAS-Manager unterscheidet sich von den cartridge-basierten Anwendungen. Bei komponenten-basierten Anwendungen wird mit Parametern auf drei Ebenen gearbeitet:

- (1) Anwendungsebene
- (2) Anwendungsinstanz-Ebene
- (3) Objekt-Ebene

¹ Siehe Kapitel 3.7.5 EJB-Anwendungen.

(1) Anwendungsebene

Die Parameter der Anwendungsebene betreffen alle Anwendungsinstanzen der Anwendung. Die Konfiguration wird somit vereinfacht, da gemeinsame Parameter aller Anwendungsinstanzen nicht für jede Anwendungsinstanz einzeln gesetzt werden müssen.

(2) Anwendungsinstanz-Ebene

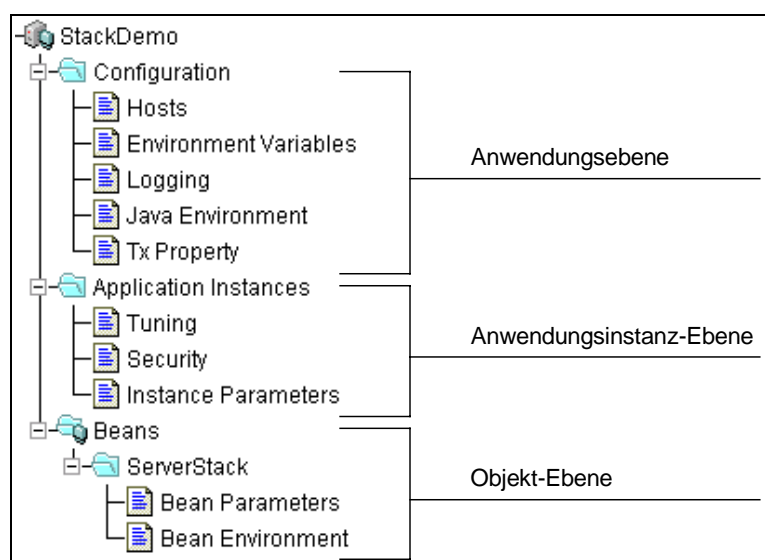
Die Parameter der Anwendungsinstanz-Ebene betreffen die Konfiguration aller Objekte einer individuellen Anwendungsinstanz.

(3) Objekt-Ebene

Die Parameter der Objekt-Ebene beziehen sich auf ein individuelles Objekt einer individuellen Anwendungsinstanz.

In Abbildung 24 wird dargestellt wie bei einer EJB-Anwendung mit dem Namen "StackDemo" diese drei Ebenen im OAS-Manager angeordnet sind.

Abbildung 24: Parameter einer EJB-Anwendung



Eine ausführlichere Beschreibung der komponenten-basierten Anwendungen erfolgt im Kapitel "3.7 Einsatzmöglichkeiten".

3.2.4 Skalierbarkeit und Lastverteilung

Wie bereits oben erwähnt, besteht die Architektur des OAS aus drei Ebenen. Aus Gründen der Flexibilität, Skalierbarkeit und Zuverlässigkeit können diese Ebenen bei einer Multi-Node-Konfiguration auf mehrere Nodes verteilt werden. Der Systemadministrator hat hierbei die Flexibilität weitere Rechner hinzuzufügen, sobald sich die Anforderungen seitens der Anwendung erhöhen.

Der OAS bietet die Fähigkeit der dynamischen Skalierbarkeit, wenn auf Anwendungen verstärkt zugegriffen wird. Durch die Funktionen des OAS wie multithreaded Cartridges¹ und Lastverteilung können Anwendungen bessere Antwortzeiten, höheren Durchsatz und maximale Performance auch bei hoher Auslastung bieten.

Zur besseren Lastverteilung (Load Balancing) bestimmt der OAS welche Cartridge-Instanz eine Anfrage bearbeiten soll. Auf diese Weise wird eine optimale Ausnutzung der verfügbaren Ressourcen gesichert. Es werden zwei Mechanismen zur Lastverteilung verwendet:

- (1) Konfigurierbare, gewichtete Lastverteilung je Anwendung
- (2) Dynamische Lastverteilung nach Systemmaßzahlen

(1) Konfigurierbare, gewichtete Lastverteilung je Anwendung

Die konfigurierbare, gewichtete Lastverteilung je Anwendung (Weighted Load Balancing) bietet die Fähigkeit, die prozentuale Anzahl von Cartridge-Instanzen einer Anwendung je Node relativ zu den anderen Nodes einer Multi-Node-Konfiguration festzulegen. Dies ist besonders nützlich, wenn der OAS als Multi-Node auf Rechnern mit verschiedener Leistungsfähigkeit installiert wird. So kann der Administrator in einer Multi-Node-Konfiguration aus z. B. drei Rechnern einen bestimmten prozentualen Anteil der Anfragen einer bestimmten Cartridge zuordnen, z. B. 30% dem ersten Node, 50% dem zweiten Node und weitere 20% dem dritten Node.

¹ Siehe Kapitel "3.2.3.1 Cartridge".

(2) Dynamische Lastverteilung nach Systemmaßzahlen

Bei der dynamischen Lastverteilung nach Systemmaßzahlen (Dynamic Load Balancing) werden bei einer Multi-Node-Konfiguration Informationen über die Auslastung eines jeden Node gemessen.

Der OAS entscheidet aufgrund dieser Informationen, auf welchen der Nodes die Anfrage bearbeitet werden soll. In diese Entscheidung fließen Hauptspeichergröße und CPU-Auslastung jedes Node mit ein.

Bei geschäftskritischen Anwendungen im WWW ist die Zuverlässigkeit der Anwendungen von großer Bedeutung. Die Architektur des OAS verhindert, dass eine einzelne Fehlerquelle den gesamten OAS zum Absturz bringen kann. Dadurch werden kostenverursachende Ausfallzeiten minimiert. Nach einem Fehler wird der Zustand vor dem Auftreten des Fehlers wiederhergestellt (Failure Recovery), unabhängig davon, ob der Fehler in einem HTTP-Listener, in einer OAS-Komponente, oder in einer Cartridge aufgetreten ist. Transaktionen werden entweder festgeschrieben, wenn sie sauber abgeschlossen sind, oder aber zurückgesetzt. Diese Fehlererkennung wird vom Resource-Manager in festgelegten Intervallen durchgeführt.

3.3 OAS-Manager

Der Oracle Application-Server Manager (OAS-Manager) ist ein graphisches Werkzeug zur Navigation durch alle Elemente einer OAS-Web-Site. Mit ihm kann der OAS administriert werden. Das heißt es können Listener, Anwendungen, Cartridges usw. hinzugefügt oder entfernt werden. Weiterhin können mit Hilfe des OAS-Manager alle diese Elemente gestartet, angehalten, konfiguriert und überwacht werden.

Der OAS-Manager wird über einen Browser aufgerufen. Dadurch besteht auch die Möglichkeit den OAS entfernt zu administrieren, also über das WWW oder ein Intranet. Als Location muss die URL der OAS-Web-Site mit dem Port 8888 eingegeben werden. In Abbildung 25 wird der OAS-Manager gezeigt.

Abbildung 25: OAS-Manager



Der OAS-Manager ist aus zwei Frames aufgebaut. In den beiden Frames wird folgendes dargestellt:

- (1) Navigationsbaum
- (2) Eigenschaften-Formular

(1) Navigationsbaum

Im linken Frame sind alle Elemente der OAS-Web-Site als Baum angeordnet. Deshalb wird diese Anordnung als Navigationsbaum bezeichnet. Der Navigationsbaum teilt die OAS-Web-Site in drei logische Ebenen ein. Diese Ebenen sind analog zu den Ebenen der Architektur des OAS¹. Dementsprechend heißen die drei Ebenen im OAS-Manager:

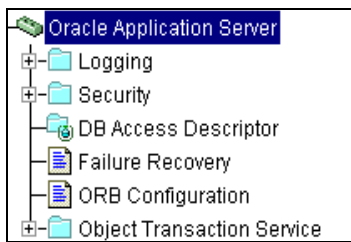
- (a) Oracle Application Server
- (b) HTTP-Listeners
- (c) Applications

(a) Oracle Application Server

Auf dieser Ebene werden Eigenschaften für das Logging, die Sicherheit, den Datenbankzugriff, den ORB und den Transaction-Service gepflegt. In Abbildung 26 wird diese Ebene im Navigationsbaum dargestellt.

¹ Siehe Kapitel "3.2 Architektur".

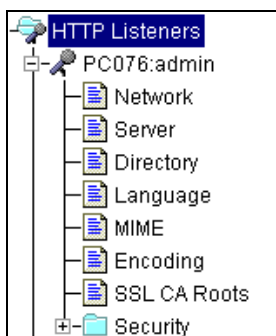
Abbildung 26: Ebene "Oracle Application Server" im Navigationsbaum



(b) HTTP-Listeners

Auf dieser Ebene werden neue HTTP-Listener erstellt und bestehende HTTP-Listener gelöscht bzw. konfiguriert. In Abbildung 27 wird diese Ebene im Navigationsbaum gezeigt.

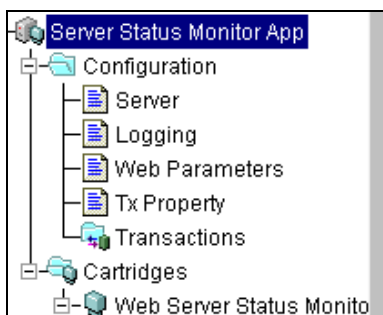
Abbildung 27: Ebene "HTTP-Listeners" im Navigationsbaum



(c) Applications

Auf dieser Ebene werden Anwendungen und Cartridges installiert, gelöscht und konfiguriert. In Abbildung 28 wird diese Ebene im Navigationsbaum dargestellt.

Abbildung 28: Ebene "Applications" im Navigationsbaum



(2) Eigenschaften-Formular

Wie in Abbildung 25 dargestellt, zeigt der rechte Frame ein Formular mit den Eigenschaften des im Navigationsbaum markierten Zweiges. In diesem Formular können die Eigenschaften angezeigt, eingetragen und geändert werden.

3.4 Datenbankzugriff

Der OAS bietet den Anwendungen die Möglichkeit über definierte Schnittstellen mit einer Datenbank zu kommunizieren und Datenbankoperationen durchzuführen. Für die verschiedenen Anwendungstypen des OAS existieren unterschiedliche Möglichkeiten auf eine Datenbank zuzugreifen.

Tabelle 5 gibt eine Übersicht über die einzelnen Anwendungstypen und ihre Möglichkeiten auf eine Datenbank zuzugreifen.

Tabelle 5: Datenbankzugriff der verschiedenen Anwendungstypen

Anwendungstyp	Datenbankzugriff
PL/SQL	DAD
JServlet	JDBC oder pl2java
C	DAD
LiveHTML	Über ICX zu einer PL/SQL-Cartridge
EJB	JDBC oder pl2java
ODBC	ODBC

Im Folgenden werden die in Tabelle 5 genannten Möglichkeiten des Datenbankzugriff anhand von Beispielen erläutert.

Database Access Descriptor (DAD)

Für den Zugriff auf Oracle7- und Oracle8-Datenbanken stellt der OAS den Database Access Descriptor (DAD) zur Verfügung. Ein DAD ist eine zentrale Ansammlung von Informationen, die der OAS benötigt, um eine Verbindung zu einer Oracle-Datenbank herzustellen. Hierdurch kann der Zugriff auf Datenbanken zentral verwaltet werden und die Verbindungsparameter müssen nicht in jeder Cartridge erneut angegeben werden.

Ein DAD kann im OAS-Manager auf OAS-Ebene über den Zweig "DB Access Descriptor" im Navigationsbaum angelegt werden. Hierfür wird der Name des DAD, der Benutzername / Passwort zur Anmeldung an der Datenbank, die IP-Adresse des DBMS-Server und der Connect-String für die Datenbank angegeben.

In Abbildung 29 wird das Formular im OAS-Manager gezeigt, in das diese Daten einzutragen sind.

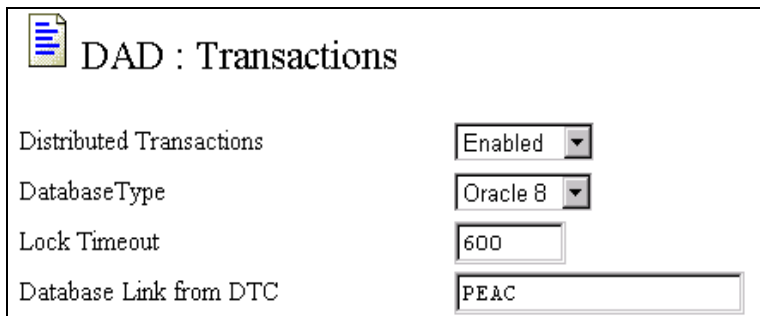
Abbildung 29: Anlegen eines DAD

Database Access Descriptor	
DAD Name	DAD_PEAC
Database User	MZE
Database User Password	***
Confirm Password	***
Database Location (Hostname)	192.168.5.179
Database Name (ORACLE_SID)	1521
Connect String	PEAC
<input type="checkbox"/> Create Database User	
<input type="checkbox"/> Change Database User Password or Tablespace	
<input checked="" type="checkbox"/> Store the username and password in the DAD	
Apply Revert Advanced Transactions Help	

Nachdem die Eingaben bestätigt wurden, wird der DAD angelegt und steht danach allen Anwendungen zur Verfügung.

Um in Anwendungen Transaktionen¹ durchführen zu können, die einen DAD als Datenbankverbindung benutzen, muss dieser DAD transaktionsfähig sein. Die erforderlichen Einstellungen für einen transaktionsfähigen DAD werden im Formular eingetragen, das Abbildung 30 darstellt. Dieses Formular wird durch Drücken des Button "Transactions" geöffnet.

Abbildung 30: Transaktionsfähiger DAD

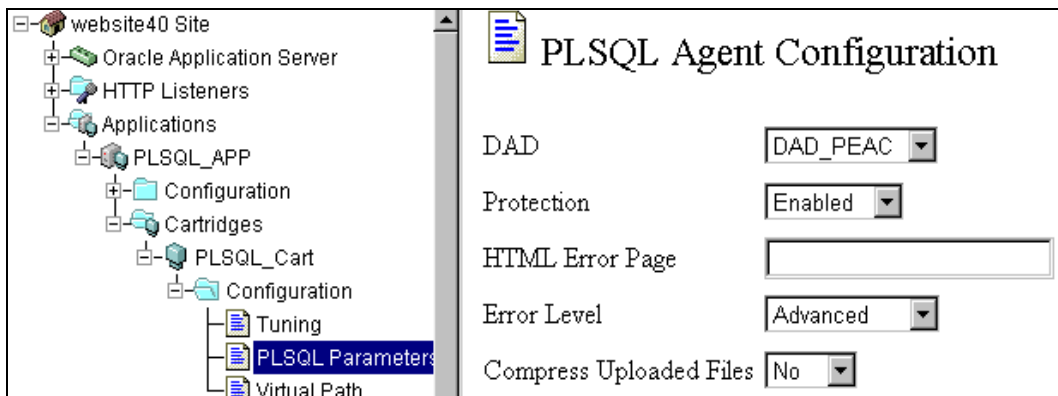


DAD : Transactions

Distributed Transactions	Enabled
Database Type	Oracle 8
Lock Timeout	600
Database Link from DTC	PEAC

Der angelegte DAD kann in den Cartridges einer Anwendungen als Datenbankverbindung angegeben werden. Als Beispiel hierfür soll eine PL/SQL-Anwendung den in Abbildung 29 angelegten DAD namens "DAD_PEAC" verwenden. Nachdem die entsprechende PL/SQL-Anwendung und deren Cartridges über den OAS-Manager angelegt worden sind, kann der DAD über den Zweig "PLSQL Parameters" im Navigationsbaum ausgewählt werden (siehe Abbildung 31).

Abbildung 31: Auswählen eines DAD für eine PL/SQL-Cartridge



PLSQL Agent Configuration

DAD	DAD_PEAC
Protection	Enabled
HTML Error Page	
Error Level	Advanced
Compress Uploaded Files	No

¹ Siehe Kapitel "3.5.3 Transaktion".

Die entsprechende Cartridge verwendet hiernach für den Datenbankzugriff den angegebenen DAD. Dabei ist zu beachten, dass in einer Anwendung die vorhandenen Cartridges verschiedene DADs verwenden können.

Java Database Connectivity (JDBC)

In Java-Anwendungen besteht die Möglichkeit für den Datenbankzugriff die Java Database Connectivity (JDBC) zu verwenden. JDBC ist ein Java-API, das Klassen, Interfaces und Exceptions bereitstellt, um SQL-Anweisungen an ein DBMS zu senden und auszuwerten. JDBC ermöglicht den Zugriff auf alle Datenbanken, die einen entsprechenden JDBC-Treiber zur Verfügung stellen. Für den Datenbankzugriff muss der entsprechende JDBC-Treiber auf dem OAS installiert sein und von der Umgebungsvariable "CLASSPATH" referenziert werden.

In Abbildung 32 wird anhand eines Codefragmentes der Aufbau einer Verbindung zu einer Oracle-Datenbank mit JDBC dargestellt.

Abbildung 32: Aufbau einer JDBC-Datenbankverbindung

```
import java.sql;
...

try {
    // 1. Oracle JDBC Treiber laden bzw. dem Driver Manager
    // bekannt machen
    Class.forName("oracle.jdbc.driver.OracleDriver");

    //2. Verbindung herstellen
    Connection connection = DriverManager.getConnection
        ("jdbc:oracle:thin:user/pwd@DB_Host:DB_Port:DB_SID");

    //3. Datenbankoperationen durchführen
    ...

    //4. Datenbankverbindung schließen
    connection.close();
}

catch ...
```

Nachdem der entsprechende JDBC-Treiber der Treiber-Verwaltung (Driver Manager) mit der Methode "Class.forName" bekannt gegeben worden ist, wird eine Datenbankverbindung mit der Methode "getConnection" hergestellt.

Neben dem zu verwendenden Treiber "jdbc:oracle:thin" wird hierbei der Benutzername, das Passwort und Datenbank-Parameter wie Host, Port und SID übergeben. Mit dieser Datenbankverbindung können die gewünschten Operationen in der entsprechenden Datenbank durchgeführt werden.

Weiterhin ist es möglich aus einer Java-Anwendung über JDBC einen DAD für den Datenbankzugriff zu verwenden. In Abbildung 33 wird der Verbindungsaufbau zu einer Datenbank gezeigt, wobei über JDBC ein DAD angesprochen wird.

Abbildung 33: Datenbankverbindung über JDBC zu einem DAD

```
import java.sql;

...

try {
    //1. Oracle JDBC-Treiber laden bzw. dem Driver Manager
    //bekannt machen
    Class.forName("oracle.jdbc.driver.OracleDriver");

    //2. Verbindung herstellen
    Connection connection =
        DriverManager.getConnection("jdbc:oracle:thin:@DAD_Name");

    //3. Datenbankoperationen durchführen
    ...

    //4. Datenbankverbindung schließen
    connection.close();
}

catch
{
    ...
}
```

Zusätzlich ermöglicht die JDBC-ODBC-Bridge den Zugriff auf eine Datenbank von einer Java-Anwendung. Auf dem Application-Server muss ein entsprechender Open Database Connectivity (ODBC) -Treiber installiert sein. Dabei ist zu beachten, dass bei der

Verwendung der JDBC-ODBC-Bridge mit erheblichen Einbußen in der Performance zu rechnen ist

pl2java

Der OAS stellt dem Entwickler das Dienstprogramm "pl2java" zur Verfügung. Mit Hilfe des Dienstprogrammes "pl2java" ist es möglich, in einer Datenbank gespeicherte PL/SQL-Programmeinheiten (Prozeduren und Funktionen) aus einer Java-Anwendung heraus aufzurufen und eventuelle Rückgabewerte an die aufrufende Java-Anwendung zurückzugeben. Dazu müssen diese PL/SQL-Programmeinheiten in einem DB-Package enthalten sein.

Für jedes in der Datenbank gespeicherte Package, aus dem Prozeduren oder Funktionen von einer Java-Anwendung aufgerufen werden sollen, wird mit dem Dienstprogramm "pl2java" eine sogenannte Wrapper-Klasse in Java generiert. Für jede Prozedur und Funktion in diesem Package wird in der Wrapper-Klasse eine Methode generiert. D. h. Packages werden auf Wrapper-Klassen und Prozeduren bzw. Funktionen auf Methoden dieser Wrapper-Klasse abgebildet.

Zusammen mit "pl2java", stehen noch weitere Java-Klassen zur Verfügung. Mit Hilfe dieser Java-Klassen können die Datentypen der Rückgabewerte von PL/SQL-Funktionen in Java-Datentypen umgewandelt werden.

Als Beispiel soll eine in der Datenbank gespeicherte PL/SQL-Funktion "ermittle_KundenNr" in einer Java-Anwendung aufgerufen werden.

Abbildung 34: PL/SQL-Package-Spezifikation und generierte Wrapper-Klasse

PL/SQL:

```
package Kunden is
  function ermittle_KundenNr(KundenName in varchar2)
    return number;
end;
```

Java Wrapper-Klasse:

```
public class Kunden {  
    public PDouble ermittle_KundenNr(PStringBuffer KundenName);  
}
```

Die PL/SQL-Funktion ist in dem DB-Package "Kunden" enthalten, dessen Spezifikation im oberen Teil der Abbildung 34 dargestellt wird. Mit "pl2java" wird für dieses Package eine Wrapper-Klasse in Java generiert, die im unteren Teil der Abbildung 34 dargestellt wird.

Um diese PL/SQL-Funktion über die Wrapper-Klasse aufrufen zu können, muss vorher eine Datenbankverbindung zur Oracle-Datenbank hergestellt werden, die das entsprechende Package speichert. In Abbildung 35 wird gezeigt, wie diese Datenbankverbindung aufgebaut wird und wie die Wrapper-Klasse für den Aufruf der Methode benutzt wird.

Abbildung 35: Aufruf einer PL/SQL-Funktion aus Java

```
// neues Session-Objekt erstellen  
Session session = new Session();  
  
// Verbinden mit der Datenbank "KundenDB" als Benutzer "Scott"  
// und dem Passwort "tiger"  
session.logon("scott", "tiger", "KundenDB");  
  
// Erstellen eines Objektes der Wrapper-Klasse "Kunden"  
Kunden kunde = new Kunden (session);  
  
// Aufrufen der PL/SQL-Funktion mit Übergabe des Kundennamen und  
// Rückgabe der Kunden-Nr.  
PDouble pKundenNr = kunde.ermittle_KundenNr(pKundenName);  
  
System.out.println("Kunde hat Kunden-Nr.: " + pKundenNr);  
  
// Datenbankverbindung schließen  
session.logoff();
```

Mit Hilfe eines Objektes der Klasse "Session" wird eine Datenbankverbindung mit der Methode "logon" aufgebaut. Als Übergabeparameter werden Benutzername, Passwort

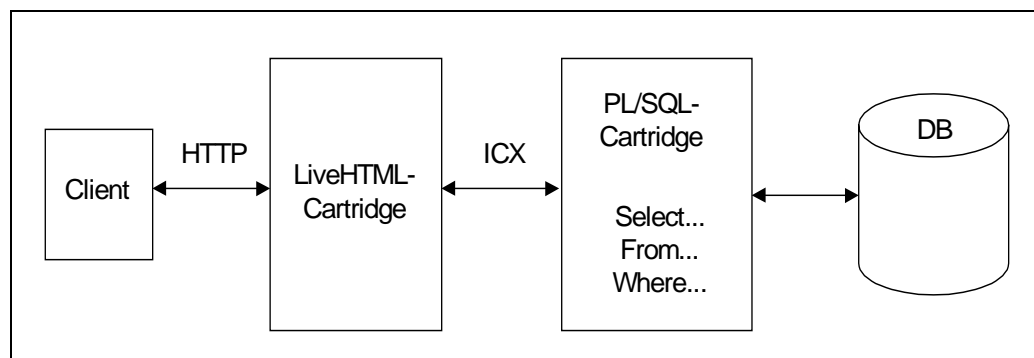
und Datenbankname übergeben. Danach wird ein Objekt der Wrapper-Klasse "Kunden" erzeugt. Mit Hilfe dieses Objektes wird die PL/SQL-Funktion "ermittle_KundenNr" aufgerufen. Diese gibt die Kunden-Nr. für den entsprechenden Kunden zurück. Nachdem das Ergebnis ausgegeben worden ist, wird die Verbindung zur Datenbank mit der Anweisung "logoff" geschlossen.

ICX (Inter Cartridge Exchange)

Inter Cartridge Exchange (ICX) ist ein Dienst, der es einer Cartridge ermöglicht, die Dienste einer anderen Cartridge in Anspruch zu nehmen. Die Cartridge kann dabei vom gleichen oder von einem anderen Typ sein.

Wie in Tabelle 5 erwähnt, kann ICX dazu verwendet werden, einen Datenbankzugriff aus einer LiveHTML-Anwendung durchzuführen. Hierfür wird aus der LiveHTML-Anwendung über ICX auf eine PL/SQL-Cartridge zugegriffen. In Abbildung 36 wird dieser Vorgang dargestellt.

Abbildung 36: Datenbankverbindung über ICX



Die PL/SQL-Cartridge stellt die Datenbankverbindung über den ihr zugeordneten DAD her, führt die Datenbankoperationen aus und gibt das Ergebnis zurück.

In Abbildung 37 wird die Anweisung gezeigt, die aus einer LiveHTML-Anwendung über ICX eine PL/SQL-Cartridge aufruft. Die PL/SQL-Cartridge ruft die in der Datenbank gespeicherte PL/SQL-Funktion "zeige_alle_Kunden" auf.

Abbildung 37: ICX-Aufruf einer PL/SQL-Cartridge aus einer LiveHTML-Anwendung

```
<!--#request url="/db/plsql/zeige_alle_Kunden"-->
```

ODBC

Open Database Connectivity (ODBC) ist ein Standard für den Datenbankzugriff, der von Microsoft entwickelt wurde. Der Datenbankzugriff über ODBC wird durch eine Cartridge realisiert, die ODBC-Cartridge. Zu beachten ist, dass die ODBC-Cartridge nur mit der Enterprise Edition des OAS ausgeliefert wird. Bei der Installation der Enterprise Edition des OAS wird die ODBC-Cartridge automatisch installiert. Die ODBC-Treiber für den Datenbankzugriff sind in der ODBC-Cartridge enthalten. Somit stehen ODBC-Treiber für den Zugriff auf folgende Datenbanken zur Verfügung:

- Oracle Server
- Sybase SQL Server
- INFORMIX-OnLine Dynamic Server
- INFORMIX-SE Server
- Microsoft SQL Server

Über die ODBC-Cartridge können drei verschiedene Arten von Anfragen an die Datenbank gestellt werden. Diese Anfragearten werden in der URL der Anfrage spezifiziert. Tabelle 6 bietet eine Übersicht über diese Anfragearten.

Tabelle 6: Anfragearten an die ODBC-Cartridge

Anfrageart	Beschreibung
ExecuteMode	Führt die übergebene SQL-Anweisung ohne Antwort an den Client aus.
TablePrint	Gibt das Ergebnis der SQL-Anweisung in Form einer HTML-Tabelle zurück.

Anfrageart	Beschreibung
StringPrint	Gibt das Ergebnis der SQL-Anweisung in Form eines Strings zurück.

Quelle: In Anlehnung an: Oracle Application Server Documentation, Release 4.0.8.1:
Developers Guide: PL/SQL and ODBC Applications, S. 11-3.

Wie die URL einer Anfrage an die ODBC-Cartridge mit Übergabe der Anfrageart als Parameter aufgebaut ist, stellt Abbildung 38 dar.

Abbildung 38: URL-Format einer Anfrage an die ODBC-Cartridge

```
http://host:port/odbc/Anfrageart?dsn=Anfragetext
```

Wobei "host:port" den OAS und den Port definiert, "odbc" für den virtuellen Pfad der ODBC-Cartridge steht und der Anfragetext die Datenbank und die SQL-Anweisung definiert. In Abbildung 39 wird ein Beispiel für eine Anfrage an die ODBC-Cartridge gezeigt.

Abbildung 39: URL-Beispiel einer Anfrage an die ODBC-Cartridge

```
http://pc060:8888/odbc/TablePrint?dsn=KundenDB&username=scott  
&password=tiger&sql=select+kundenname+from+kunden
```

Das Beispiel zeigt eine Anfrage mit der Anfrageart "TablePrint" an die ODBC-Datenquelle (dsn = data source name) "KundenDB" mit dem Benutzer "scott", dem Passwort "tiger" und einer Select-Anweisung.

3.5 Interaktionsmodelle

Es gibt verschiedene Interaktionsmodelle für die Interaktion zwischen Client und Server. Die vom OAS unterstützten Interaktionsmodelle sind:

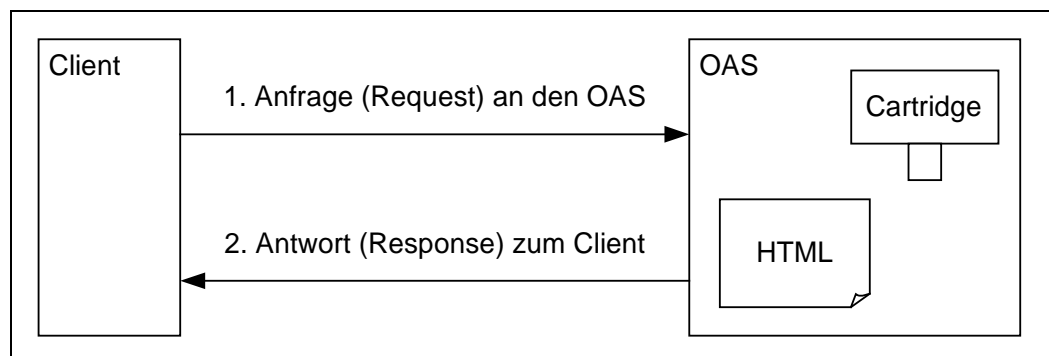
- Anfrage/Antwort-Interaktion
- Sitzung
- Transaktion

In den nachfolgenden Kapiteln werden diese Interaktionsmodelle erläutert. Weiterhin wird dargestellt, wie der OAS diese Interaktionsmodelle unterstützt.

3.5.1 Anfrage/Antwort-Interaktion

Die klassische Interaktion zwischen Client und OAS ist die Anfrage/Antwort-Interaktion über das HTTP, die in Abbildung 40 dargestellt wird.

Abbildung 40: Anfrage/Antwort-Modell



Der Client sendet eine Anfrage an den OAS. Der OAS sendet die entsprechende Antwort in Form eines statischen oder dynamisch generierten HTML-Dokumentes zurück an den Client.

Anfragen, die sich an eine Cartridge richten, leitet der Dispatcher an eine Cartridge des entsprechenden Typs weiter. Die nächste verfügbare Cartridge-Instanz bearbeitet die Anfrage und die Antwort wird an den Client zurückgesendet.

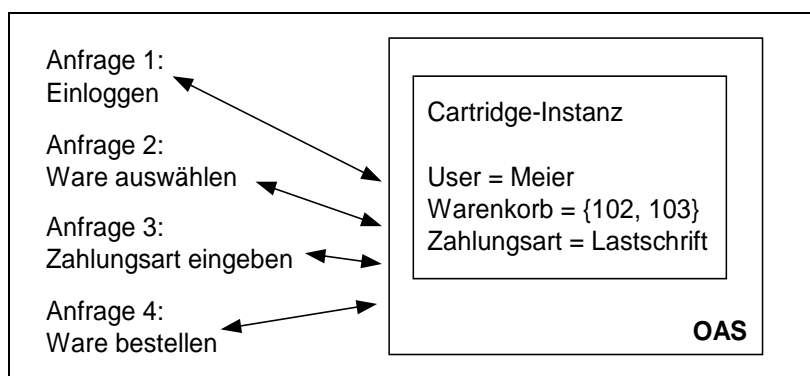
Nach der Bearbeitung der Anfrage wird die logische Verbindung zwischen dem Client-Browser und dem OAS abgebrochen. Kontextinformationen innerhalb dieser Kommunikation gehen mit der Verbindung verloren. Deshalb wird diese Art der Kommunikation auch als zustandslose Kommunikation bezeichnet.

3.5.2 Sitzung

Viele Anwendungsfälle erfordern, dass Informationen aus einer früheren Anfrage (Kontextinformationen) in einer neuen Anfrage zur Verfügung stehen. Hierfür muss eine dauerhafte logische Verbindung zwischen dem Client-Browser und dem OAS hergestellt werden. Diese logische Verbindung wird durch eine Sitzung (Session) realisiert. Bei einer Sitzung werden die Kontextinformationen auf dem Server gespeichert und stehen über die gesamte Sitzung, d. h. über mehrere Anfragen zur Verfügung. In dieser Interaktion bleibt die Sitzung solange bestehen, bis ein bestimmtes Zeitlimit erreicht ist, oder die Verbindung von einem der beiden Kommunikationspartner beendet wird.

In Abbildung 41 stellt ein Client in einer Sitzung mehrere zusammenhängende Anfragen an den OAS, um einen Bestellvorgang durchzuführen.

Abbildung 41: Mehrere Anfragen in einer Sitzung



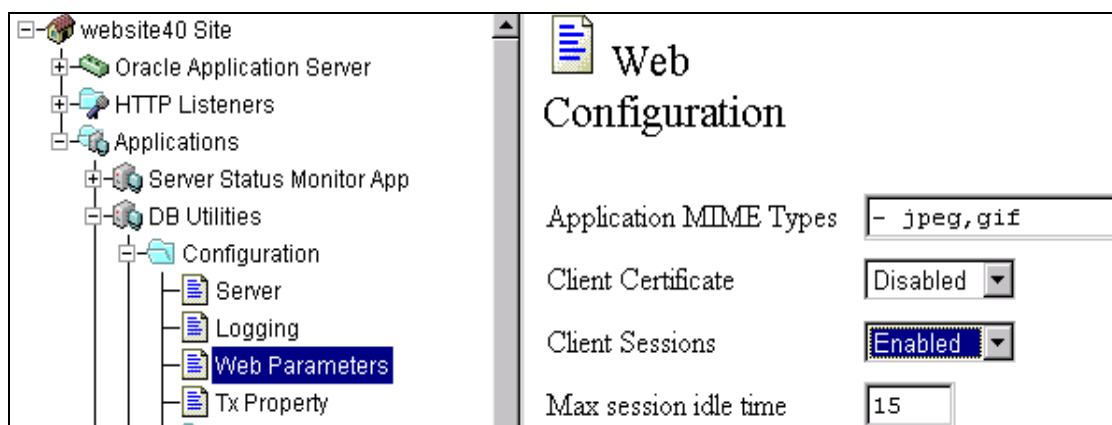
Nachdem der Benutzer (Meier) sich eingeloggt hat, Ware (Artikel 102 und Artikel 103) ausgesucht und die gewünschte Zahlungsart (Lastschrift) angegeben hat, bestellt er die Ware.

Mit der ersten Anfrage loggt sich der Benutzer in die Anwendung ein. Damit wird eine Sitzung begonnen und diese einer bestimmten Cartridge-Instanz zugeordnet. Die Kontextinformationen, hier der Benutzername (User), werden auf dem OAS in der Cartridge-Instanz gespeichert. Alle folgenden Anfragen dieses Client werden dieser Sitzung bzw. dieser Cartridge-Instanz zugeordnet und können auf vorhergegangene Anfragen dieses Clients zurückgreifen.

Durch Speichern der Kontextinformationen auf dem OAS können der Kommunikation zwischen Client-Browser und OAS Zustände zugeordnet werden. In Abbildung 41 können der Kommunikation die Stati "Benutzer eingeloggt", "Ware ausgewählt", "Zahlungsart angegeben" und "Ware bestellt" zugeordnet werden. Der OAS hat somit die Möglichkeit zu erkennen, in welchen Zustand sich die Kommunikation momentan befindet.

Mit Hilfe des "Session-Service", den der OAS anbietet, können für cartridge-basierte Anwendungen Sitzungen realisiert werden. Dazu muss im OAS-Manager im Formular "Web Configuration" der entsprechenden Anwendung der Parameter "Client Sessions" aktiviert werden. In Abbildung 42 wird die Einstellung dieses Parameters bei der Anwendung "DB Utilities" dargestellt.

Abbildung 42: Aktivieren einer Sitzung im OAS-Manager



Die Aktivierung des Parameters "Client Sessions" in einer Anwendung signalisiert dem Dispatcher, dass die Cartridges dieser Anwendung innerhalb einer Sitzung ausgeführt

werden sollen. Dazu leitet der Dispatcher eingehende Anfragen eines Client immer an die zugeordnete Cartridge-Instanz weiter. Zusätzlich kann eine Zeitspanne "Max session idle time" definiert werden, nach der die Sitzung automatisch beendet wird, wenn keine weiteren Anfragen vom entsprechenden Client eingehen.

Bei komponenten-basierten Anwendungen ist es möglich Sitzungen programmtechnisch mittels bestimmter Session-APIs zu realisieren.

3.5.3 Transaktion

Unkritische und wiederholbare Interaktionen können entweder als Anfrage/Antwort-Interaktion oder als Sitzung durchgeführt werden. Mehrere Geschäftsvorgänge, die zusammen einen Geschäftsprozess darstellen, sollten dagegen unbedingt als Transaktion durchgeführt werden. Eine Transaktion ist eine logische Einheit mehrerer Operationen, die Änderungen an vorhandenen Datenbeständen vornehmen. Hauptmerkmal einer Transaktion ist, dass entweder alle oder keine der enthaltenen Operationen durchgeführt werden.

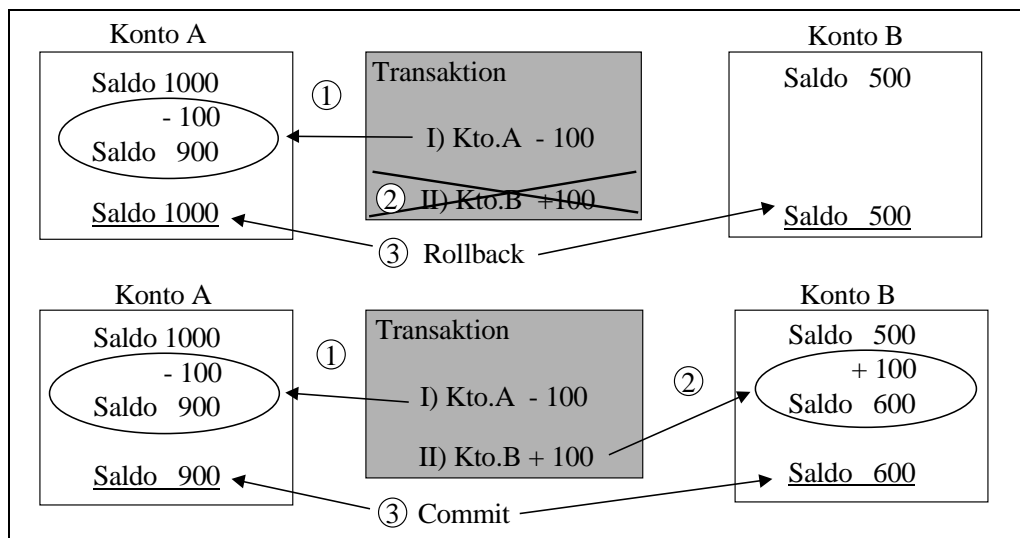
Ein klassisches Beispiel für einen Geschäftsprozess ist eine Banküberweisung. Sie besteht insgesamt aus zwei Geschäftsvorgängen: ein Konto wird um einen bestimmten Betrag vermindert, ein anderes um genau diesen Betrag erhöht.

In Abbildung 43 wird die Banküberweisung "Überweise 100 von Konto A nach Konto B" gezeigt, die in einer Transaktion durchgeführt wird. Die Banküberweisung ist nur dann erfolgreich, wenn Geschäftsvorgang I (vermindere Konto A um 100) und Geschäftsvorgang II (erhöhe Konto B um 100) erfolgreich durchgeführt wurden. Sollte ein Geschäftsvorgang scheitern, so kann der Geschäftsprozess nicht erfolgreich abgeschlossen werden und der bereits durchgeführte Geschäftsvorgang muss zurückgerollt werden, damit die Konten korrekte Salden aufweisen.

Im oberen Teil der Abbildung 43 ist eine gescheiterte Transaktion dargestellt. Nachdem Geschäftsvorgang I erfolgreich durchgeführt wurde, scheitert Geschäftsvorgang II. Da die Transaktion nicht erfolgreich durchgeführt werden konnte, muss der Ausgangszustand

wiederhergestellt werden. Dazu muss der durchgeführte Geschäftsvorgang I zurückgerollt werden (Rollback). Nachdem ein Rollback durchgeführt worden ist, befinden sich beide Salden in der Ausgangssituation.

Abbildung 43: Banküberweisung als Transaktion



Im unteren Teil der Abbildung 43 wird dagegen eine erfolgreiche Banküberweisung gezeigt. Die beiden Geschäftsvorgänge I und II werden hier erfolgreich durchgeführt und vermindern bzw. erhöhen das jeweilige Konto um den entsprechenden Betrag. Die Änderungen der Konten werden endgültig durch eine Bestätigung festgeschrieben (Commit) und beide Konten weisen den korrekten Saldo aus.

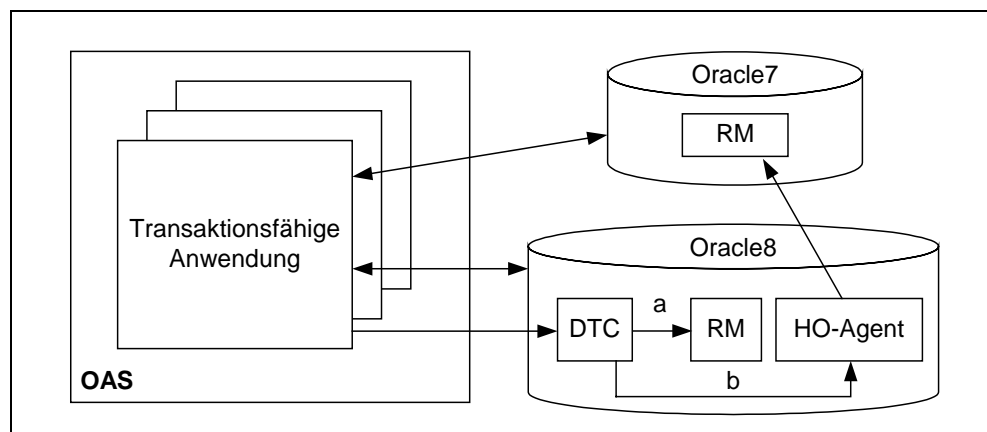
Der OAS stellt für die Realisierung von Transaktionen den "Transaction-Service" bereit. Der Transaction-Service ist nur in der Enterprise Edition des OAS verfügbar. Er stellt Dienste zur Verfügung, damit mehrere Objekte (Anwendungen und Datenbanken) an einer Transaktion beteiligt sein können.

Der Transaction-Service unterstützt auch verteilte Transaktionen. Das heißt die beteiligten Objekte einer Transaktion müssen nicht auf demselben Rechner vorhanden sein, sondern können auch auf mehrere Rechner verteilt sein.

Für die Abwicklung einer Transaktion sind im wesentlichen vier Komponenten verantwortlich, die in Abbildung 44 dargestellt werden. Diese vier Komponenten sind die folgenden:

- (1) Transaktionsfähige Anwendung
- (2) Distributed Transaction Coordinator (DTC)
- (3) Resource-Manager (RM)
- (4) Heterogeneous OTS (HO) -Agent

Abbildung 44: Komponenten einer Transaktion



Quelle: In Anlehnung an: Oracle Application Server Documentation, Release 4.0.8.1: Administration Guide, S. 11-4.

(1) Transaktionsfähige Anwendung

Eine transaktionsfähige Anwendung führt Operationen innerhalb einer Transaktion aus. Diese Anwendung ist verantwortlich für den Beginn einer Transaktion (Begin), für den Verbindungsaufbau und -abbau zu allen Datenbanken und für die Entscheidung, ob eine Transaktion durchgeführt (Commit) oder zurückgerollt (Rollback) werden soll.

(2) Distributed Transaction Coordinator (DTC)

Der Distributed Transaction Coordinator (DTC) koordiniert die erforderlichen Operationen, um Änderungen in der Datenbank festzuschreiben, die in einer Transaktion durchgeführt wurden.

Sollte bei der Ausführung der Transaktion ein Fehler auftreten, so weist der DTC den Resource-Manager (RM) an, alle Änderungen der Transaktion zurückzurollen. In Abbildung 44 wird dies durch den Pfeil "a" angezeigt. Der DTC muss von einer Oracle8-Datenbank bereitgestellt werden.

(3) Resource-Manager (RM)

Der Resource-Manager (RM) ist eine zentrale Stelle in der Datenbank, in der alle innerhalb einer Transaktion durchgeführten Änderungen aufgezeichnet werden. Dem OAS präsentiert sich der Resource-Manager als transaktionsfähiger DAD, der eine Oracle7- oder Oracle8-Datenbank anbindet. Eine Anwendung kann mehrere RM bzw. Oracle-Datenbanken in einer Transaktion benutzen. Die Verwaltung dieser verteilten Transaktion wird ebenfalls durch den DTC übernommen.

(4) Heterogeneous OTS (HO) -Agent

Ist eine Oracle7-Datenbank in die verteilte Transaktion einbezogen, so muss ein HO-Agent als Übersetzer eingesetzt werden. Der HO-Agent übersetzt dabei die Befehle des DTC der Oracle8-Datenbank in eine Form, die vom RM der Oracle7-Datenbank verstanden wird. In Abbildung 44 wird dies durch den Pfeil "b" angezeigt.

Der OAS bietet dem Entwickler zwei Möglichkeiten den Transaction-Service in einer Anwendung zu nutzen. Es werden folgende Transaktionen unterstützt:

- Deklarative Transaktion
- Programmtechnische Transaktion

Diese Möglichkeiten den Transaction-Service in einer Anwendung zu nutzen, werden in den folgenden beiden Kapiteln beschrieben. Zuvor jedoch gibt Tabelle 7 einen Überblick, wie Transaktionen bei den wichtigsten Anwendungstypen definiert werden können.

Tabelle 7: Definieren von Transaktionen bei den Anwendungstypen

Anwendungstyp	Definieren der Transaktionen
PL/SQL	Deklarativ durch Angabe einer URI
JServlet	Programmtechnisch mit JTS
EJB	Programmtechnisch mit JTS oder deklarativ im Deployment-Descriptor

Quelle: Oracle Application Server Documentation, Release 4.0.8.1: Administration Guide, S. 11-4.

3.5.3.1 Deklarative Transaktion

Eine deklarative Transaktion wird abhängig vom Anwendungstyp entweder im OAS-Manager oder im Deployment-Descriptor eines Objektes deklariert. Zur Realisierung von deklarativen Transaktionen müssen keine Änderungen in der Anwendungslogik vorgenommen werden.

Der OAS unterstützt für bestimmte Anwendungstypen die folgenden deklarativen Transaktionen.

- (1) Web-deklarative Transaktion
- (2) Deployment-deklarative Transaktion

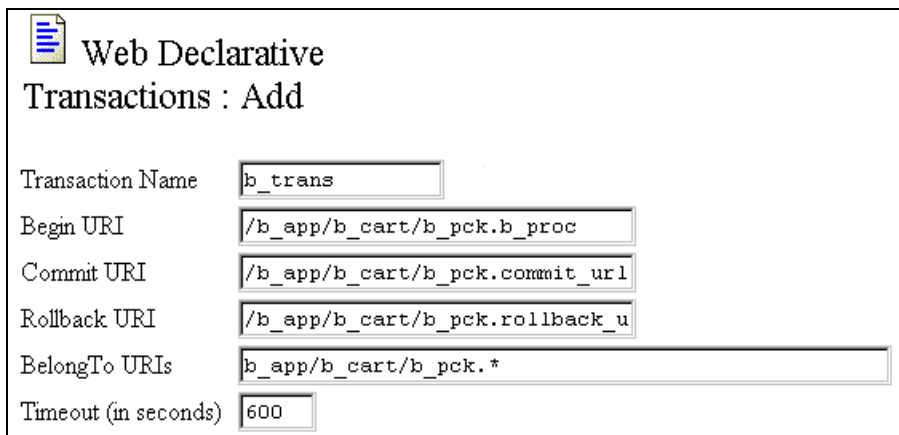
(1) Web-deklarative Transaktion

Web-deklarative Transaktionen können nur bei PL/SQL-Anwendungen verwendet werden. Sollen Datenbankoperationen innerhalb web-deklarativer Transaktionen durchgeführt werden, muss jedem Transaktionsbefehl (Begin, Commit und Rollback) eine URI zugeordnet werden. Wird während der Ausführung der Anwendung eine zugeordnete URI aufgerufen, wird automatisch der dieser URI zugeordnete Transaktionsbefehl ausgeführt.

Dazu muss der Anwendung im OAS-Manager ein transaktionsfähiger DAD zugeordnet werden.

Im Folgenden betrachten wir eine PL/SQL-Anwendung, die eine deklarative Transaktion verwendet. In dieser Transaktion wird eine Stored-Procedure mit dem Namen "b_proc" ausgeführt. Zuerst wird ein transaktionsfähiger DAD angelegt, der die Transaktionsbefehle ausführt. Hiernach werden im OAS-Manager auf Anwendungsebene im Zweig "Transactions" die URIs für die einzelnen Transaktionsbefehle festgelegt, wie in Abbildung 45 gezeigt. Die URIs sind dabei frei wählbar, sollten sich allerdings der Übersicht halber an eine sprechende Namenskonvention halten. Die URIs in Abbildung 45 folgen der Namenskonvention: /<Anwendung>/<Cartridge>/<Package>.<Proc>.

Abbildung 45: Definieren einer web-deklarativen Transaktion



Web Declarative Transactions : Add	
Transaction Name	<input type="text" value="b_trans"/>
Begin URI	<input type="text" value="/b_app/b_cart/b_pck.b_proc"/>
Commit URI	<input type="text" value="/b_app/b_cart/b_pck.commit_url"/>
Rollback URI	<input type="text" value="/b_app/b_cart/b_pck.rollback_u"/>
BelongTo URIs	<input type="text" value="b_app/b_cart/b_pck.*"/>
Timeout (in seconds)	<input type="text" value="600"/>

Wurde eine Transaktion durch den Aufruf der "Begin URI" begonnen, so werden alle Anfragen, die in den Zuständigkeitsbereich dieser Transaktion fallen, auch innerhalb dieser Transaktion durchgeführt. Der Zuständigkeitsbereich wird im Feld "BelongTo URIs" definiert. Beim Aufruf der "Commit URI" bzw. "Rollback URI" wird die Transaktion festgeschrieben bzw. zurückgerollt. Werden nach dem Beginn einer Transaktion Anfragen gestellt, die nicht in den Zuständigkeitsbereich dieser Transaktion fallen, so werden die von dieser Anfrage durchgeführten Änderungen auch nicht von der Transaktion bei den Befehlen Commit bzw. Rollback festgeschrieben bzw. zurückgerollt.

Der Parameter "Timeout" definiert die Zeit, die verstreichen muss, bis eine aktionslose Transaktion mit Rollback beendet wird.

(2) Deployment-deklarative Transaktion

Deployment-deklarative Transaktionen können nur bei EJB-Anwendungen verwendet werden. Sollen Datenbankoperationen innerhalb deployment-deklarativer Transaktionen durchgeführt werden, muss für die entsprechende Anwendung im OAS-Manager ein transaktionsfähiger DAD angegeben werden. Zusätzlich wird für jedes Objekt in dessen Deployment-Descriptor definiert, in welchem Transaktionsmodus dieses Objekt ausgeführt werden soll. Ein Überblick dieser Transaktionsmodi wird in Tabelle 8 dargestellt.

Tabelle 8: Transaktionsmodi bei der deployment-deklarativen Transaktion

Transaktionsmodus	Beschreibung
TX_REQUIRED	Wird eine Methode dieses Objektes aus einer Transaktion aufgerufen, wird sie in derselben Transaktion ausgeführt. Wird eine Methode dieses Objektes aus keiner Transaktion aufgerufen, so wird für sie eine neue Transaktion begonnen.
TX_NOT_SUPPORTED	Diese Methode wird immer außerhalb einer Transaktion ausgeführt, unabhängig von einer anderen Transaktion.
TX_SUPPORTS	Wird eine Methode dieses Objektes aus einer Transaktion aufgerufen, wird sie in derselben Transaktion ausgeführt. Wird eine Methode dieses Objektes aus keiner Transaktion aufgerufen, wird für sie keine neue Transaktion begonnen.
TX_REQUIRES_NEW	Methoden mit diesem Transaktionsmodus werden immer in einer neuen Transaktion ausgeführt, unabhängig von einer anderen Transaktion. Nachdem die Methode beendet ist, wird automatisch die neue Transaktion mit Rollback oder Commit beendet.
TX_MANDATORY	Alle Methoden dieses Objektes müssen aus einer Transaktion heraus aufgerufen werden, sonst wird die Exception "TransactionRequired" zurückgegeben.
TX_BEAN_MANAGED	Transaktionen werden programmtechnisch durchgeführt. Wird eine Methode dieses Objektes aus einer Transaktion aufgerufen, wird sie immer außerhalb dieser Transaktion ausgeführt.

Quelle: Oracle Application Server Documentation, Release 4.0.8.1: Developers Guide: EJB and CORBA Applications, S. 6-6.

In der Anwendungslogik muss ein JDBC-Treiber für den Datenbankzugriff angegeben werden. Dieser stellt dann eine Verbindung zu einem transaktionsfähigen DAD her.

Das folgende Beispiel einer Überweisung soll die deployment-deklarative Transaktion verdeutlichen. Ein EJB-Objekt "bank" enthält die Methoden "ueberweisen", "einzahlen" und "auszahlen". Ein weiteres EJB-Objekt "protokoll" enthält die Methode "protokoll_schreiben".

Innerhalb der Methode "ueberweisen" werden die drei Methoden "auszahlen", "einzahlen" und "protokoll_schreiben" nacheinander aufgerufen. Die Überweisung kann nur erfolgreich durchgeführt werden, wenn die beiden Methoden "auszahlen" und "einzahlen" erfolgreich durchgeführt wurden. Sollte eine der beiden Methoden scheitern, so ist auch die Überweisung gescheitert und die Ausgangssituation muss durch ein Rollback der bereits durchgeführten Änderungen wiederhergestellt werden. Unabhängig davon, ob die Überweisung erfolgreich oder gescheitert ist soll ein Protokoll mit der Methode "protokoll_schreiben" erstellt werden.

In Abbildung 46 und Abbildung 47 wird dargestellt, wie die Transaktionsmodi in den Deployment-Descriptors der beiden EJB-Objekte definiert werden:

Abbildung 46: Deployment-Descriptor des EJB-Objektes "bank"

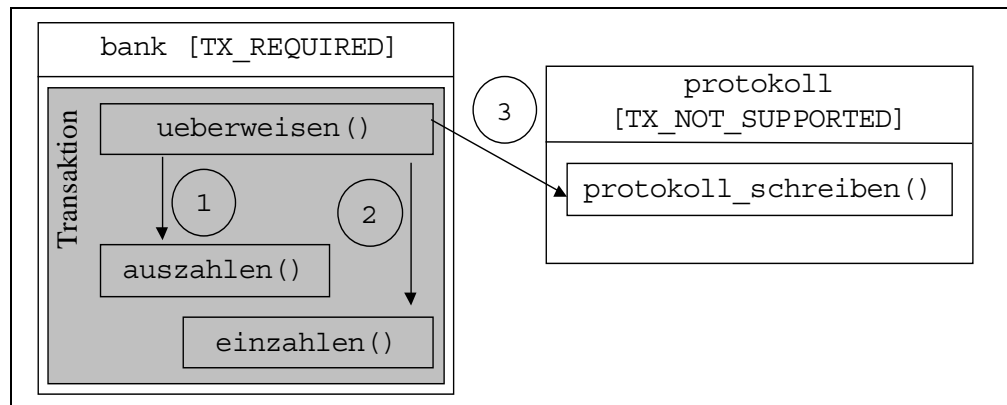
```
[bank]
...
transactionMode = TX_REQUIRED
...
```

Abbildung 47: Deployment-Descriptor des EJB-Objektes "protokoll"

```
[protokoll]
...
transactionMode = TX_NOT_SUPPORTED
...
```

Für das Objekt "bank" ist der Transaktionsmodus "TX_REQUIRED" definiert. Deshalb werden, wie in Abbildung 48 gezeigt, die Methoden "ueberweisen", "auszahlen" und "einzahlen" innerhalb derselben Transaktion ausgeführt.

Abbildung 48: Transaktionsbereich einer Anwendung



Da für das Objekt "protokoll" der Transaktionsmodus "TX_NOT_SUPPORTED" definiert worden ist, wird die Methode "protokoll_schreiben" außerhalb der Transaktion ausgeführt. Die Methode "protokoll_schreiben" wird also immer ausgeführt, egal ob die Überweisung erfolgreich oder gescheitert ist.

In Abbildung 49 wird der Quellcode der beiden Klassen "Bank" und "Protokoll" gezeigt, von denen die Objekte "bank" und "protokoll" erzeugt werden.

Abbildung 49: Quellcode der Klassen "Bank" und "Protokoll"

```

public class Bank implements javax.ejb.SessionBean {

    // DB-URL mit JTS-fähigem JDBC-Treiber und Angabe des
    // transaktionsfähigen DAD
    url = "jdbc:oracle:jts7:@dadName;

    // Beim Aufruf dieser Methode wird automatisch eine neue
    // Transaktion begonnen
    public void ueberweisen() {

        Protokoll protokoll = new Protokoll();

        Connection conn = DriverManager.getConnection(url);

        // Datenbankoperationen ausführen

```

```
        PreparedStatement pst = conn.create...
        pst.executePrepared(...);

        auszahlen();
        einzahlen();
        protokoll.protokoll_schreiben();
        conn.close(); //DB-Verbindung schließen
    }

    // Diese Methode setzt die, in der Methode ueberweisen()
    // begonnene Transaktion fort
    public einzahlen() {

        Connection conn = DriverManager.getConnection(url);

        // Datenbankoperationen ausführen
        PreparedStatement pst = conn.create...
        pst.executePrepared(...);
        ...
        conn.close(); // DB-Verbindung schließen
    }

    // Diese Methode setzt die, in der Methode ueberweisen()
    // begonnene Transaktion fort
    public auszahlen() {

        Connection conn = DriverManager.getConnection(url);

        // Datenbankoperationen ausführen
        PreparedStatement pst = conn.create...
        pst.executePrepared(...);
        ...
        conn.close(); // DB-Verbindung schließen
    }
}

public class Protokoll implements javax.ejb.SessionBean {

    // Diese Methode wird außerhalb der in der Methode
    // ueberweisen() begonnenen Transaktion ausgeführt
    public void protokoll_schreiben() {
        ...
    }
}
```

3.5.3.2 Programmtechnische Transaktion

Programmtechnische Transaktionen können nur in den Anwendungstypen JServlet und EJB realisiert werden. Sollen Datenbankoperationen innerhalb programmtechnischer Transaktionen durchgeführt werden, müssen die Transaktionsbefehle Begin, Commit und

Rollback in der Anwendungslogik implementiert werden. Weiterhin muss ein transaktionsfähiger DAD vorhanden sein oder angelegt werden.

Java stellt mit dem Java Transaction Service (JTS) Methoden zur Verfügung, um diese Transaktionsbefehle in einer Anwendung implementieren zu können. JTS verwendet für den Datenbankzugriff JDBC. Daher muss bei der Verwendung von JTS in der Anwendungslogik ein JTS-fähiger JDBC-Treiber für den Datenbankzugriff angegeben werden. Dieser stellt dann eine Verbindung zu einem transaktionsfähigen DAD her.

In Abbildung 50 wird ein Codefragment einer EJB-Anwendung dargestellt, das eine programmtechnische Transaktion¹ unter Verwendung des JTS implementiert.

Abbildung 50: Codefragment einer programmtechnischen Transaktion mit JTS

```
public class Kunde implements javax.ejb.SessionBean {  
    UserTransaction usertran = null;  
  
    public void kunde_einfuegen() {  
        // DB-URL mit JTS-fähigem JDBC-Treiber und Angabe des  
        // transaktionsfähigen DAD  
        url = "jdbc:oracle:jts7:@dadName;  
  
        Connection conn = DriverManager.getConnection(url);  
        usertran.begin();           //Beginn der Transaktion  
        PreparedStatement pst = conn.create... //Kundendaten einfügen  
        pst.executePrepared(...);  
        ...  
        adresse_einfuegen(); //Adresse des Kunden einfügen  
        usertran.commit();       //Transaktion beenden  
        conn.close();          //DB-Verbindung schließen  
    }  
  
    public void adresse_einfuegen() {  
        Connection conn = DriverManager.getConnection(...);  
  
        // überprüfen ob bereits eine Transaktion besteht  
        boolean transaktionNichtVorhanden =  
            (usertran.getStatus() == usertran.STATUS_NO_TRANSACTION);  
  
        // Wenn Transaktion nicht vorhanden neue öffnen, sonst  
        // die bestehende benutzen  
        if (transaktionNichtVorhanden)
```

¹ Die Transaktionsbefehle sind fett gedruckt.

```
        usertran.begin();

        PreparedStatement pst = conn.create... // Adressdaten
        pst.executePrepared(...);             // einfügen
        ...
        // Transaktion abschließen, wenn in dieser Methode begonnen
        if (transaktionNichtVorhanden)
            usertran.commit();

        conn.close(); // DB-Verbindung schließen
    }
}
```

Die Klasse "Kunde" besteht aus den zwei Methoden "kunde_einfuegen" und "adresse_einfuegen". Wird ein neuer Kunde mit der Methode "kunde_einfuegen" erfasst, so wird zusätzlich eine neue Adresse mit der Methode "adresse_einfuegen" eingefügt. Beide Methoden sollen dazu in einer Transaktion ausgeführt werden.

Es soll allerdings auch möglich sein Adressen unabhängig von Kunden einfügen zu können, d. h. die Methode "adresse_einfuegen" soll auch separat aufrufbar sein.

Die Datenbankoperationen der Methode "adresse_einfuegen" sollen dabei entweder in der von der Methode "kunde_einfuegen" begonnenen Transaktion, oder in einer neuen Transaktion durchgeführt werden.

Im Quellcode der Abbildung 50 wird zuerst eine Variable "usertran" vom Typ "UserTransaction" deklariert, die eine Transaktion repräsentiert. In der Methode "kunde_einfuegen" wird die Transaktion mit der Methode "usertran.begin" begonnen. Hiernach werden die Datenbankoperationen durchgeführt und die Methode "adresse_einfuegen" aufgerufen. Zum Schluss der Methode werden die Änderungen der Transaktion mit der Anweisung "usertran.commit" in der Datenbank festgeschrieben.

In der Methode "adresse_einfuegen" wird mit der Bedingung "usertran.getStatus() == usertran.STATUS_NO_TRANSACTION" überprüft, ob bereits eine Transaktion begonnen wurde. Das Ergebnis wird einer boolschen Variablen zugewiesen. Wurde bereits eine Transaktion begonnen, so wird diese für das Festschreiben von Änderungen der folgenden Datenbankoperationen verwendet. Wurde keine

Transaktion begonnen, so beginnt die Methode eine neue Transaktion. Diese schreibt nach dem Durchführen der Datenbankoperationen die Änderungen fest.

3.6 Sicherheit

In diesem Kapitel werden die Sicherheitsaspekte erläutert, die der OAS bietet. Die beiden wesentlichen Sicherheitsaspekte des OAS sind:

- Authentifizierung
- Datenverschlüsselung

Auf diese beiden Sicherheitsaspekte wird in den beiden nachfolgenden Kapiteln ausführlich eingegangen. Zusätzlich wird in einem weiteren Kapitel speziell auf die Sicherheit bei PL/SQL-Anwendungen eingegangen.

3.6.1 Authentifizierung

Authentifizierung ist eine Überprüfung, ob der Client bzw. Server auch wirklich der ist, der er behauptet zu sein. Der OAS bietet zur Authentifizierung verschiedene Authentifizierungsschemata¹, die berechtigte Benutzer bzw. Rechner definieren und die Authentifizierung durchführen. Für den Begriff Authentifizierungsschema wird hier auch abkürzend der Begriff Schema verwendet. Zur Authentifizierung benutzt der OAS seine Komponente "Auth-Server".

Auth-Server

Der Auth-Server führt Authentifizierungen im Auftrag eines Dispatchers oder des RM-Proxy durch. Dazu überprüft der Auth-Server, ob ein Benutzer oder Rechner dazu berechtigt ist, eine bestimmte Anfrage auszuführen.

¹ Siehe "Benutzer-Authentifizierungsschema" und "Host-Authentifizierungsschema".

Die Authentifizierungsschemata werden im Auth-Server selbst implementiert, genauer gesagt in den Auth-Providern. Der Auth-Server besteht demnach aus den beiden Bestandteilen:

- (1) Auth-Broker
- (2) Auth-Provider

(1) Auth-Broker

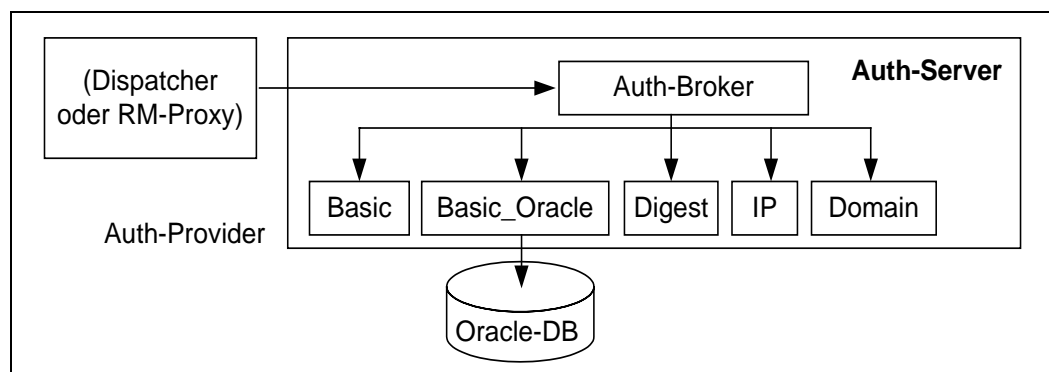
Der Auth-Broker empfängt Authentifizierungsanfragen von einem Dispatcher oder dem RM-Proxy und gibt diese an den entsprechenden Auth-Provider weiter. Der Auth-Broker hat also nur die Aufgabe die Authentifizierungsanfragen zu koordinieren, nicht aber die Authentifizierung durchzuführen.

(2) Auth-Provider

Die Auth-Provider implementieren die Authentifizierungsschemata und führen die Authentifizierung durch. Der Auth-Broker hat zu jedem Authentifizierungsschema einen Auth-Provider.

In Abbildung 51 wird gezeigt, wie der Auth-Broker eine Authentifizierungsanfrage an den entsprechenden Auth-Provider weiterleitet.

Abbildung 51: Auth-Broker und Auth-Provider



Quelle: In Anlehnung an: Oracle Application Server Documentation, Release 4.0.8.1: Security Guide, S. 2-3.

Eine Authentifizierungsanfrage, die ein Dispatcher oder der RM-Proxy an den Auth-Server sendet, enthält einen "Authentication-String", der das zu verwendende Authentifizierungsschemata genau spezifiziert. Diese Authentifizierungsanfrage wird zuerst an den Auth-Broker gegeben, der sie seinerseits an den entsprechenden Auth-Provider zur Bearbeitung vermittelt. Der Auth-Provider gibt dann die Antwort zurück an den Auth-Broker. Der Auth-Broker gibt die Antwort an den Dispatcher bzw RM-Proxy.

Der Auth-Server kann in zwei Modi ausgeführt werden. Die Funktionsweise ist in beiden Modi die gleiche, der Unterschied liegt in der Performance und der Unterstützung einer verteilten Sicherheitsarchitektur. Die Modi, in denen ein Auth-Server ausgeführt werden kann, sind:

- (1) Inmemory-Mode
- (2) ORB-Mode

(1) Inmemory-Mode

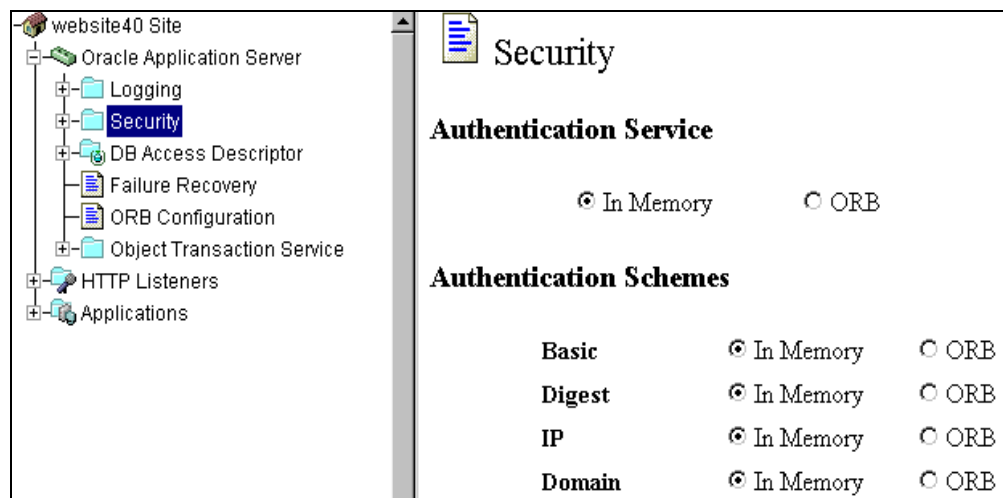
Im Inmemory-Mode existiert für jeden Client, der den Auth-Server anspricht, eine eigene Instanz des Auth-Broker und / oder der Auth-Provider. Hierbei muss der Auth-Server auf demselben Rechner ausgeführt werden, wie die Komponenten die ihn aufrufen. Dieser Mode benötigt mehr Speicher, ist aber schneller als der ORB-Mode.

(2) ORB-Mode

Im ORB-Mode existiert genau eine Instanz des Auth-Broker und / oder der Auth-Provider. Hierbei muss der Auth-Server nicht auf demselben Rechner ausgeführt werden, wie die Komponenten die ihn aufrufen. Dieser Mode unterstützt eine verteilte Sicherheitsarchitektur, benötigt weniger Speicher und ist etwas langsamer als der Inmemory-Mode.

Wie Abbildung 52 dargestellt, werden die Modi für den Auth-Server mit Hilfe des OAS-Manager gesetzt.

Abbildung 52: Definieren der Modi für Auth-Broker und Auth-Provider



Die Modi können auch kombiniert werden, indem für Auth-Broker und Auth-Provider verschiedene Modi gesetzt werden.

Benutzer-Authentifizierungsschema

Die Aufgabe eines Benutzer-Authentifizierungsschemas (User Authentication Scheme) ist es, einen Benutzer zu identifizieren. Der Benutzer erhält ein Login-Fenster, mit dem er sich einloggen muss.

Über dieses Authentifizierungsschema können Cartridges oder Komponenten geschützt werden.

Versucht ein Benutzer auf eine geschützte Anwendung zuzugreifen, wird der HTTP-Status 401 an den Browser zurückgegeben, der diesen dazu veranlasst, ein Dialogfenster zum Einloggen einzublenden.

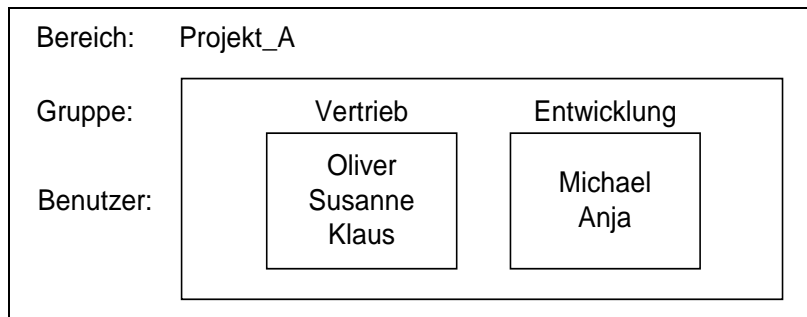
Der OAS 4.0.8.1 bietet die folgenden Benutzer-Authentifizierungsschemata:

- (1) Basic
- (2) Digest
- (3) Basic_Oracle
- (4) Crypt

(1) Basic

Im Schema "Basic" werden Bereiche (realms), Gruppen (groups) und Benutzer (user) definiert. Jeder Benutzer gehört zu einer oder mehreren Gruppen und jede Gruppe zu einem oder mehreren Bereichen. In Abbildung 53 werden diese Zusammenhänge gezeigt.

Abbildung 53: Bereich, Gruppe und Benutzer bei Basic und Digest¹



Die Benutzer "Oliver", "Susanne" und "Klaus" gehören zur Gruppe "Vertrieb" und die Benutzer "Michael" und "Anja" zur Gruppe "Entwicklung". Beide Gruppen gehören zum Bereich "Projekt_A". Die definierten Bereiche, Gruppen und Benutzer werden in einer Konfigurationsdatei auf dem OAS gespeichert. Die Passwörter werden in dieser Datei verschlüsselt gespeichert.

Um den Zugriff auf eine Anwendung nur definierten Benutzern zu gewähren, kann einer Cartridge bzw. Komponente ein Bereich zugeordnet werden. Greift ein Benutzer auf diese Cartridge bzw. Komponente zu, so wird er aufgefordert einen Benutzernamen und ein Passwort anzugeben. Um die Authentifizierung erfolgreich durchzuführen muss der Benutzer sich mit einem im entsprechenden Bereich definierten Benutzer anmelden. Der Benutzername und das Passwort werden bei diesem Schema verschlüsselt übertragen.

(2) Digest

Das Schema "Digest" unterscheidet sich vom Schema Basic nur in der Übertragung von Benutzernamen und Passwort vom Client zum Server. Beim Schema Digest basiert die

¹ Siehe Abschnitt (2) Digest.

Übertragung von Benutzername und Passwort auf einem Prinzip das sich "challenge/response" nennt. Hierbei generiert der Server einen Zwischenwert und sendet diesen zum Client. Der Client sendet eine Checksumme, berechnet aus Benutzername, Passwort, Zwischenwert, HTTP-Methode und der angefragten URI, zum Server. Dadurch wird die Übertragung von Benutzername und Passwort gesichert.

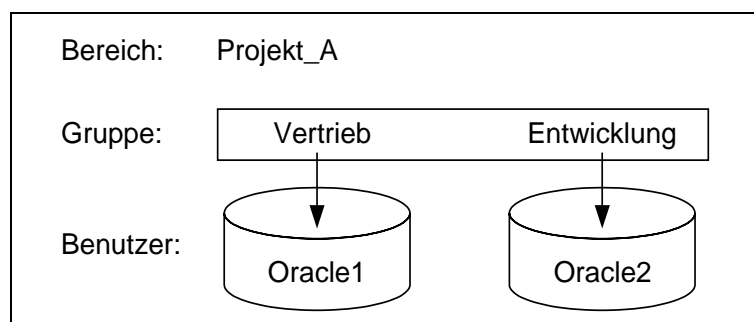
Das Digest Schema wird von wenigen Browsern unterstützt, so z. B. nicht von Netscape und Microsoft. Greift ein Browser, der das Schema Digest nicht unterstützt, auf eine Cartridge oder Komponente zu, die einem Digest Schema zugeordnet ist, so schaltet der Listener automatisch auf das Schema Basic um.

(3) Basic_Oracle

Ähnlich wie bei den Schemata Basic und Digest wird beim Schema "Basic_Oracle" mit Bereichen, Gruppen und Benutzer gearbeitet. Der Unterschied besteht darin, dass bei Basic_Oracle auf dem OAS keine Benutzer definiert werden, sondern nur Gruppen. Jeder Gruppe wird eine Oracle-Datenbank zugeordnet, und damit alle Benutzer der Oracle-Datenbank.

In Abbildung 54 sei eine Anwendung durch den Bereich "Projekt_A" geschützt. Ein Benutzer hat nur dann Zugriff auf diese Anwendung, wenn er einen gültigen Benutzernamen / Passwort für eine der Datenbanken "Oracle1" oder "Oracle2" angibt.

Abbildung 54: Bereich, Gruppe und Benutzer bei Basic_Oracle



Dieses Schema hat den Vorteil, dass nicht zwei Listen von Benutzern verwaltet werden müssen (auf dem OAS und in der Datenbank), sondern nur die Benutzer der Datenbank.

(4) Crypt

Das Schema "Crypt" wird verwendet, um den Zugriff auf Cartridges zu kontrollieren, die eine UNIX-spezifische Verschlüsselung von Passwörtern verwenden. Hierbei wird der Benutzername und das Passwort aus einer benutzerdefinierten Datei gelesen.

Dieses Schema wird nur aus Gründen der Vollständigkeit genannt, es wird hier nicht näher darauf eingegangen.

Host-Authentifizierungsschema

Mit Hilfe eines Host-Authentifizierungsschemas (Host Authentication Scheme) besteht die Möglichkeit eine Liste von IP-Adressen oder Domain-Namen zu definieren, die auf Cartridges bzw. Komponenten zugreifen dürfen, bzw. nicht zugreifen dürfen.

Greift ein Client auf eine Anwendung zu, die mit einem Host-Authentifizierungsschema geschützt ist, so prüft der OAS die IP-Adresse bzw. den Domain-Namen des anfragenden Rechners. Die Anfrage wird nur bearbeitet, wenn der Client eine Zugriffsberechtigung auf die geschützte Anwendung hat. Der OAS 4.0.8.1 unterstützt die folgenden Host-Authentifizierungsschemata:

- (1) IP
- (2) Domain

(1) IP

Beim Schema IP basiert die Authentifizierung auf der IP-Adresse des anfragenden Rechners.

(2) Domain

Beim Schema Domain basiert die Authentifizierung auf dem Domain-Namen des anfragenden Rechners.

In beiden Schemata kennzeichnet ein "+" vor einer IP-Adresse oder einem Domain-Namen, dass diesem Rechner der Zugriff auf die entsprechende Anwendung gewährt wird. Ein "-" vor der IP-Adresse oder dem Domain-Namen kennzeichnet, dass diesem Rechner der Zugriff auf die entsprechende Anwendung verweigert wird.

Zusätzlich kann das Zeichen "*" als Wildcard in IP-Adressen oder Domain-Namen verwendet werden. Tabelle 9 zeigt einige Beispiele.

Tabelle 9: Beispiele für Host-Authentifizierungsschemata

Beispiel	Beschreibung
+144.25.10.100	Dem Rechner mit dieser IP-Adresse wird der Zugriff gewährt.
-144.66.44.111	Dem Rechner mit dieser IP-Adresse wird der Zugriff verweigert.
-140.84.4.*	Rechnern, deren IP-Adresse mit 140.84.4. beginnt, wird der Zugriff verweigert.
+pc60.opitz-partner.de	Dem Rechner mit diesem Domain-Namen wird der Zugriff gewährt.
+140.84.4.*	Rechnern, deren IP-Adresse mit 140.84.4. beginnt, wird der Zugriff gewährt.
-* .opitz-partner.de	Rechnern der Domain opitz-partner.de wird der Zugriff verweigert.

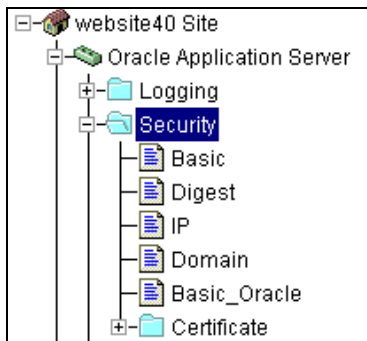
Quelle: In Anlehnung an: Oracle Application Server Documentation, Release 4.0.8.1: Security Guide, S. 2-11.

In beiden Schemata werden Gruppen definiert, und diesen Gruppen ein oder mehrere Muster zugeordnet, wie sie in Tabelle 9 dargestellt werden. So kann z. B. eine Gruppe "mygroup" definiert werden und ihr eine Liste von Mustern zugeordnet werden, die z. B. aus "+*.inmygroup.com" und "-notinmygroup.com" bestehen kann.

Implizit wird an jede Liste ein "-" angehängt, damit dem anfragenden Rechner kein Zugriff auf die geschützte Anwendung gewährt wird, wenn dessen IP-Adresse oder Domain-Name auf kein Muster in der Liste zutrifft.

Die Benutzer-Authentifizierungsschemata und Host-Authentifizierungsschemata werden mit Hilfe des OAS-Manager verwaltet (siehe Abbildung 55).

Abbildung 55: Authentifizierungsschemata im OAS-Manager

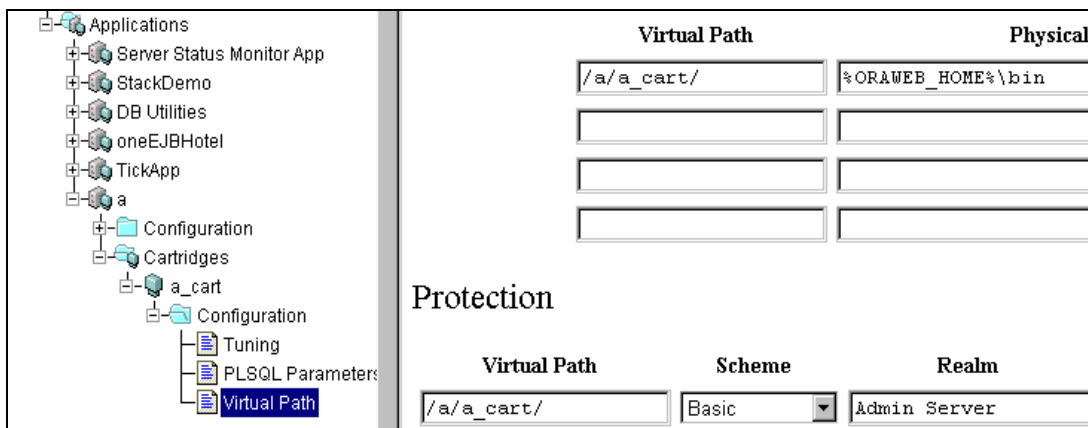


Konfigurieren der Anwendungen zur Authentifizierung

Cartridge-basierte Anwendung:

Bei der Konfiguration einer Cartridge wird dem virtuellen Pfad, über den die Cartridge angesprochen wird, ein Authentifizierungsschema zugeordnet (siehe Abbildung 56).

Abbildung 56: Zuordnung eines virtuellen Pfades zu einem Schema



Wird eine Anfrage an die Cartridge gestellt, gibt der Virtual-Path-Manager dem Dispatcher einen Authentication-String zurück, der angibt welches Schema zur Authentifizierung benutzt werden soll. Der Dispatcher stellt mit diesem Authentication-String eine Authentifizierungsanfrage an den Auth-Server. Daraufhin führt der Auth-Server die Authentifizierung durch.

Komponenten-basierte Anwendung:

Bei der Konfiguration der Anwendungsinstanzen wird explizit ein "Authentication-String" definiert, der angibt welches Authentifizierungsschema benutzt werden soll. In Abbildung 57 wird das Formular "Security" der Anwendung "StackDemo" gezeigt, in dem der Authentication-String anzugeben ist.

Abbildung 57: Formular zur Angabe des Authentication-String



Werden mehrere Schemata zugeordnet, so müssen diese durch UND ("&") bzw. ODER ("|") verknüpft werden. In Abbildung 58 wird das Format und ein Beispiel für einen Authentication-String dargestellt.

Abbildung 58: Authentication-String: Format und Beispiel

Format:

<Schema>(<Bereich>) [{"|"| "&"} <Schema> (<Bereich>) ...]

Bsp.:

IP(Bereich1) & Basic(Bereich2)

Das Beispiel zeigt eine UND-Verknüpfung aus einem Host-Authentifizierungsschema und einem Benutzer-Authentifizierungsschema. Nur Rechner mit IP-Adressen aus Bereich1 haben Zugriff auf die Anwendung und der Benutzer muss sich mit einem Benutzer aus Bereich2 einloggen.

Wird eine Anfrage an die Anwendung gestellt, gibt die Anwendung dem RM-Proxy einen Authentication-String zurück, der angibt welches Schema zur Authentifizierung benutzt

werden soll. Der RM-Proxy stellt mit diesem Authentication-String eine Authentifizierungsanfrage an den Auth-Server. Daraufhin führt der Auth-Server die Authentifizierung durch.

3.6.2 Datenverschlüsselung

Der OAS bietet die Möglichkeit, Daten die zwischen Client und OAS, sowie zwischen den OAS-Komponenten übertragen werden, mit dem Public-Key-Verschlüsselungsverfahren Secure Sockets Layer (SSL) zu verschlüsseln. Bei der Datenverschlüsselung mit SSL wird das Protokoll HTTPS verwendet. SSL ist ein Protokoll der Transport-Schicht und hat folgende Funktionen:

- (1) Datenschutz
- (2) Datenintegrität
- (3) Authentifizierung

(1) Datenschutz

Unter Datenschutz ist zu verstehen, dass niemand außer dem beabsichtigten Empfänger die Daten lesen kann. SSL unterstützt den Datenschutz, indem es die Daten verschlüsselt, die zwischen Client und Server übertragen werden.

Für die Verschlüsselung mit SSL wird ein Schlüsselpaar benötigt, das aus einem Public-Key (Öffentlicher Schlüssel) und einem Private-Key (Privater Schlüssel) besteht.

Die Funktionsweise hierbei ist folgendermaßen: Der Public-Key einer Institution A wird anderen Institutionen zur Verfügung gestellt. Senden diese Institutionen Daten an Institution A, wird der Public-Key benutzt, um die Daten für eine sichere Übertragung zu verschlüsseln. Auf Seite der Institution A werden die Daten mit Hilfe des Private-Key entschlüsselt. Da nur Institution A den Private-Key besitzt, kann keiner außer Institution A die Daten entschlüsseln. Diese Verschlüsselung funktioniert analog in umgekehrter Richtung, wobei die Daten bei der Institution A mit dem Private-Key verschlüsselt werden und nur von den Institutionen entschlüsselt werden können, die den Public-Key besitzen.

Der Public-Key muss zertifiziert werden, damit der Sender der Daten auch sicher sein kann, dass der Public-Key auch wirklich von der Institution stammt, zu der er Daten senden will. Ausgegeben werden diese Zertifikate von einer Zertifizierungsstelle (Certificate authority = CA). Eine CA ist eine Einrichtung, die Zertifikate an Einzelpersonen oder Unternehmen nur dann ausgibt, wenn es die Einzelperson bzw. das Unternehmen überprüft hat. Ein Zertifikat enthält folgende Informationen:

- Eine eindeutige Seriennummer, die dem Zertifikat von der CA zugeordnet wird.
- Ein Kennzeichen für den Algorithmus, der verwendet wurde, um das Zertifikat zu signieren.
- Den Namen der CA, die das Zertifikat ausgegeben hat.
- Einen Zeitraum für welchen das Zertifikat gültig ist.
- Den Namen des Benutzers (der Institution) des Zertifikats.
- Den Public-Key der Institution.
- Die Signatur der CA.

(2) Datenintegrität

Unter Datenintegrität ist zu verstehen, dass die Daten so beim Empfänger ankommen, wie sie beim Sender abgeschickt wurden. D. h. SSL schützt die Daten während der Übertragung vor Änderungen.

(3) Authentifizierung

Authentifizierung ist eine Überprüfung, ob der Client bzw. Server auch wirklich der ist, der er behauptet zu sein. Dazu werden Zertifikate¹ verwendet. Verbindet sich ein Client-Browser mit einem Server, so präsentiert der Server-Listener sein Zertifikat². Der Browser kann dann entweder das Zertifikat automatisch akzeptieren³, oder er fordert den Benutzer auf zu entscheiden, ob das Zertifikat akzeptiert werden soll.

¹ Siehe Abschnitt (1) Datenschutz.

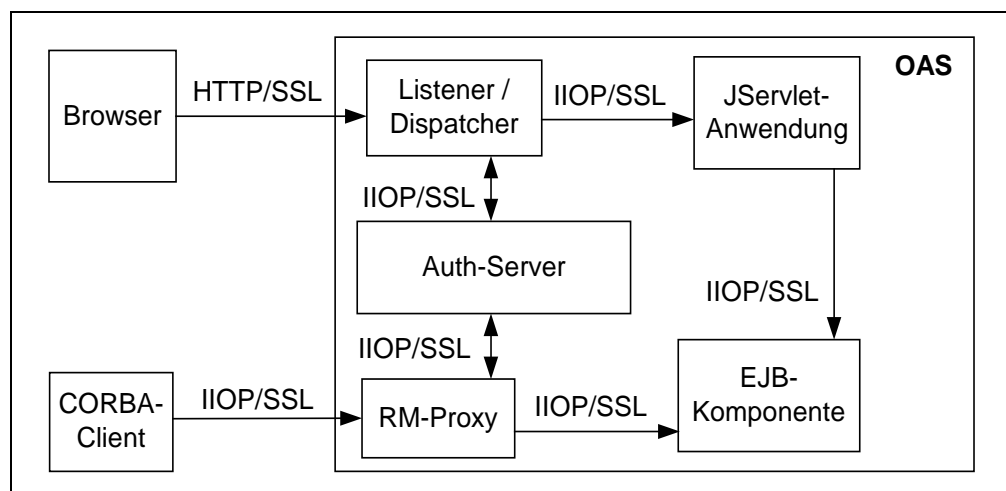
² Der Listener muss dafür speziell konfiguriert werden, damit er SSL unterstützt.

³ Dazu muss das Zertifikat in den "Site certificates" des Browsers enthalten sein.

Zusätzlich zur Authentifizierung des Servers kann der Server vom Client eine Authentifizierung fordern. Dazu muss der Client über ein eigenes Zertifikat verfügen und dieses dem Server auf Verlangen vorzeigen.

In Abbildung 59 werden die Kommunikationswege zwischen Client und OAS, sowie zwischen den OAS-Komponenten gezeigt.

Abbildung 59: Kommunikationswege des OAS



Quelle: In Anlehnung an: Oracle Application Server Documentation, Release 4.0.8.1: Security Guide, S. 1-4.

Daten die zwischen Client und OAS übertragen werden, können mit SSL (Secure Sockets Layer) verschlüsselt werden. Dies ist möglich, wenn die Clients über die Übertragungsprotokolle HTTP oder IIOP mit dem OAS kommunizieren.

Sind die OAS-Komponenten verteilt, also auf mehreren Rechnern installiert, so kommunizieren die Komponenten untereinander über IIOP und die zu übertragenden Daten können auch hier mit SSL verschlüsselt werden.

Im Folgenden werden die Schritte gezeigt die notwendig sind, um die SSL-Unterstützung eines Listeners auf dem OAS einzurichten:

- (1) Zertifikat-Anforderung generieren
- (2) Einreichen der Zertifikat-Anforderung

- (3) Zertifikat installieren
- (4) Port für SSL-verschlüsselte Daten einrichten

(1) Zertifikat-Anforderung generieren

Im Lieferumfang des OAS ist ein Dienstprogramm namens "genreq" enthalten, das dazu benutzt werden kann, eine Zertifikat-Anforderung zu generieren.

Dieses Dienstprogramm generiert ein Schlüsselpaar (Public-Key, Private-Key) und schreibt den Public-Key in eine Datei. Diese Datei kann an eine CA geschickt werden, um ein Zertifikat zu erhalten. Zudem wird eine Datei generiert, die den Private-Key enthält.

(2) Einreichen der Zertifikat-Anforderung

Nach dem Generieren der Zertifikat-Anforderung wird diese bei einer CA eingereicht, um das Zertifikat zu erhalten.

(3) Zertifikat installieren

Nach dem Erhalt des Zertifikates wird es mit Hilfe des OAS-Manager installiert. In Abbildung 60 wird das Formular dargestellt, in das die Informationen für die Installation des Zertifikates einzutragen sind.

Abbildung 60: SSL-Formular eines Listeners

Cert Label	Cert File	Dist Name File
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>
<input type="text"/>	<input type="text"/>	<input type="text"/>

Apply Revert Help

(4) Port für SSL-verschlüsselte Daten einrichten

Um die Daten, die zwischen Client-Browser und Listener übertragen werden, mit SSL zu verschlüsseln, müssen ein oder mehrere Ports für SSL-Verbindungen eingerichtet werden, auf denen der oder die Listener lauschen. Der Port 443 ist der Default-Port für SSL-Verbindungen. Gibt ein Benutzer z. B. die URL "https://..." im Browser ein, richtet sich die Anfrage automatisch an Port 443.

Sind diese Schritte durchgeführt, so werden die Daten mit SSL verschlüsselt, die zwischen Browser und konfiguriertem Listener übertragen werden.

3.6.3 Sicherheit bei PL/SQL-Anwendungen

Eine PL/SQL-Cartridge hat den Zweck mit Hilfe eines DAD¹ Prozeduren auszuführen, die in einer Oracle-Datenbank gespeichert sind. In der PL/SQL-Cartridge wird der DAD angegeben, der für Anfragen an eine Oracle-Datenbank benutzt werden soll.

Wird der DB-Benutzername und das Passwort nicht im DAD gespeichert, so führt die PL/SQL-Cartridge eine "BASIC_NEW Authentifizierung" durch. Hierbei wird der OAS angewiesen, bei jedem Aufruf der PL/SQL-Cartridge den DB-Benutzernamen und das Passwort vom Benutzer zu verlangen. Zu beachten ist, dass hierbei Benutzername und Passwort in Klartext, also unverschlüsselt übertragen werden.

Sollte eine Verschlüsselung der Daten notwendig sein, so kann im entsprechenden Listener die Verschlüsselung der zu übertragenden Daten mit SSL aktiviert werden.

Es gibt noch eine weitere Sicherheitsmethode der PL/SQL-Cartridge. Wie oben erwähnt, kann bei einer PL/SQL-Anwendung eine Prozedur, die in der Datenbank gespeichert ist, über die Eingabe einer URL im Client-Browser aufgerufen werden. Das Problem hierbei ist, dass mit dieser Art des Aufrufs alle Prozeduren in einer Datenbank aufrufbar sind und nicht nur die, die als Einstiegspunkte in eine Anwendung vorgesehen sind. Es ist also möglich, dass der Benutzer Teile der Anwendung umgehen kann, oder Prozeduren

¹ Siehe Kapitel "3.4 Datenbankzugriff".

aufrufen kann, die er eigentlich gar nicht ausführen darf, was ein erhebliches Sicherheitsrisiko darstellen kann.

Um dieses Problem zu beheben, stellt eine PL/SQL-Anwendung eine Sicherheitsmethode zur Verfügung, die sich "Protected Package" nennt. Die in diesen Protected Packages enthaltenen Prozeduren und Funktionen können nur von anderen Prozeduren aufgerufen werden und nicht direkt über eine URL. So ist der Benutzer gezwungen über die vorgesehenen Einstiegspunkte in die Anwendung einzusteigen.

3.7 Einsatzmöglichkeiten

Auf dem OAS können Anwendungen verschiedener Typen bzw. Programmiersprachen eingesetzt werden. In den folgenden Kapiteln wird auf die Einsatzmöglichkeiten der cartridge-basierten Anwendungstypen PL/SQL, Perl, LiveHTML und JServlet eingegangen, sowie auf die Einsatzmöglichkeiten der komponenten-basierten Anwendungstypen C++-CORBA und EJB.

Die Einsatzmöglichkeiten werden anhand von Beispielen beschrieben. Zum Testen wurde die OAS 4.0.8.1 Enterprise Edition in der Single-Node-Konfiguration auf Windows NT 4.0 installiert.

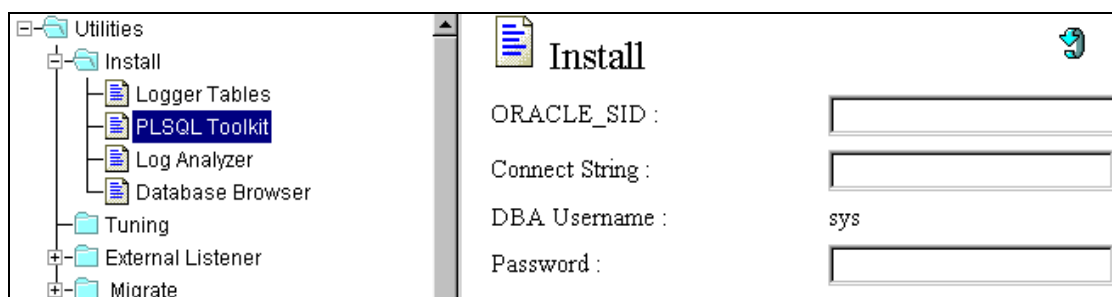
3.7.1 PL/SQL-Anwendungen

PL/SQL ist eine prozedurale Erweiterung von SQL, der Standard-Abfragesprache für relationale Datenbanken. Mit einer PL/SQL-Anwendung auf dem OAS können auf Anfrage HTML-Dokumente aus PL/SQL-Programmeinheiten, die in einer Oracle-Datenbank gespeichert sind (Stored-Procedures), generiert werden. Das hat den Vorteil, dass der Entwickler Datenbankoperationen wie z. B. "Select", "Insert", "Delete" und weitere Konstrukte wie Schleifen oder If-Anweisungen direkt in der Anwendungslogik implementieren kann.

Zur Generierung der HTML-Dokumente steht dem Entwickler das PL/SQL-WebToolkit zur Verfügung, das aus mehreren PL/SQL-Packages besteht. Mit den in diesen Packages enthaltenen Prozeduren und Funktionen ist es möglich, die Struktur eines HTML-Dokumentes zu generieren.

Das PL/SQL-WebToolkit muss vor dem Implementieren einer PL/SQL-Anwendung in der entsprechenden Oracle-Datenbank installiert werden. Dazu ist die OAS Welcome Page über den Node-Manager-Listener¹ aufzurufen und der Link "OAS-Utilities" anzuklicken. Mit dem Utility "PLSQL Toolkit" im Ordner "Install" kann das PL/SQL-WebToolkit in der Oracle-Datenbank installiert werden. Wie in Abbildung 61 dargestellt, muss dazu im Eigenschaften-Formular der Connect-String der Datenbank und das Passwort des Datenbank-Benutzers "SYS" eingegeben werden.

Abbildung 61: Installation des PL/SQL-WebToolkit



Nach erfolgreicher Installation stehen zahlreiche Packages zur Verfügung, die das Generieren von HTML-Dokumenten unterstützen. Die wichtigsten Packages werden in Tabelle 10 beschrieben.

Tabelle 10: Packages des PL/SQL-WebToolkit

Package	Beschreibung
http (Hypertext Procedures)	Enthält Prozeduren um HTML-Tags zu erzeugen. Zu jedem HTML-Tag gibt es eine korrespondierende Prozedur.

¹ Siehe Kapitel "3.2.1 HTTP-Listener-Ebene".

Package	Beschreibung
htf (Hypertext Functions)	Enthält die gleichen Programmeinheiten wie das Package <code>http</code> , nur in Form von Funktionen. Diese werden als Übergabeparameter für Prozeduren des Package <code>http</code> verwendet. Zu jedem HTML-Tag gibt es eine korrespondierende Funktion.
<code>owa_util</code>	Enthält Programmeinheiten zur Unterstützung von dynamischem SQL, zum Auslesen von CGI-Umgebungsvariablen und zur Datumsbehandlung.
<code>owa_pattern</code>	Enthält Programmeinheiten zum Vergleichen und Manipulieren von Strings.
<code>owa_cookie</code>	Enthält Programmeinheiten, die es ermöglichen Cookies an Client-Browser zu senden, oder von einem Client-Browser zu empfangen.

Quelle: Oracle Application Server Documentation, Release 4.0.8.1: Developer's Guide: PL/SQL and ODBC Applications, S. 4-4.

Im folgenden Beispiel soll eine PL/SQL-Anwendung erstellt werden, die auf Anfrage Daten aus einer Oracle-Datenbank mit einer Select-Anweisung abfragt. Auf Basis dieser Daten soll mit Hilfe des PL/SQL-WebToolkit ein HTML-Dokument generiert werden. Dieses HTML-Dokument soll eine Kundenliste mit Namen und Telefonnummern in Tabellenform anzeigen.

Die Prozedur "`liste_kunde`" in Abbildung 62 fragt die Kundendaten aus der Datenbank ab und generiert auf Basis dieser Daten die Kundenliste.

Abbildung 62: PL/SQL-Prozedur zur Generierung eines HTML-Dokumentes

```
create or replace procedure liste_kunde
is
begin

    http.htmlOpen; --erzeugt <html>
    http.headOpen; --erzeugt <head>
    http.title('Kundenliste'); --erzeugt <title>...</title>
    http.headClose; --erzeugt </head>

    http.bodyOpen; --erzeugt <body>

    -- Überschrift der Tabelle
```



```
http.header(1, 'Liste aller Datensätze der Tabelle Kunde');

-- Daten in Form einer Tabelle mit Überschrift anzeigen
http.tableOpen('border=1'); --erzeugt <table border=1>

-- Tabellenüberschrift einfügen
http.tableRowOpen; --erzeugt <tr>
  http.tableData('Kundenname'); --erzeugt <td>...</td>
  http.tableData('Telefon'); --erzeugt <td>...</td>
http.tableRowClose; --erzeugt </tr>

-- Daten abfragen und in Form einer HTML-Tabelle aufbereiten
for rec_kunde in (select Kunde_name,
                      kunde_telefon_nr
                  from kunde
                  order by kunde_name)
loop
  http.tableRowOpen; --erzeugt <tr>
    http.tableData(rec_kunde.kunde_name); --erzeugt <td>...</td>
    http.tableData(rec_kunde.kunde_telefon_nr);
  http.tableRowClose; --erzeugt </tr>
end loop;

http.tableClose; --erzeugt </table>

http.bodyClose; --erzeugt </body>
http.htmlClose; --erzeugt </html>

end;
```

Zu Beginn der Prozedur wird das HTML-Tag "<html>", das den Anfang eines HTML-Dokumentes signalisiert, mit der Prozedur "http.htmlOpen" generiert. Nachdem weitere HTML-Tags wie "<head>", "<title>" oder "<body>" generiert wurden, werden die Kundendaten mit einer Select-Anweisung aus der Datenbank abgefragt. Für die Ausgabe der Daten wird eine Tabelle mit dem Tag "<TABLE border=1>" begonnen. Die einzelnen Zeilen der Tabelle werden innerhalb einer Schleife mit den Kundendaten gefüllt. Zum Schluss wird die Tabelle, sowie das HTML-Dokument beendet.

Die PL/SQL-Prozedur "liste_kunde" wird in der Datenbank gespeichert. Im nächsten Schritt wird eine PL/SQL-Anwendung "PLSQL_APP" und eine PL/SQL-Cartridge "PLSQL_CART" mit Hilfe des OAS-Manager eingerichtet. Wie in Abbildung 31 gezeigt, muss bei der Einrichtung der PL/SQL-Cartridge neben anderen Parametern ein DAD angegeben werden. Der DAD muss dabei auf die Datenbank verweisen, in der die auszuführende PL/SQL-Prozedur gespeichert ist.

Hiernach ist die PL/SQL-Anwendung installiert und kann über einen Browser aufgerufen werden. Das Format der URL zum Aufrufen einer PL/SQL-Anwendung wird in Abbildung 63 dargestellt.

Abbildung 63: URL-Format zum Aufrufen einer PL/SQL-Anwendung

```
http://<OASHost>/<appname>/<cartname>/<stored_procedure>
```

In Abbildung 64 wird die URL zum Ausführen der für das Beispiel angelegten Cartridge "PLSQL_CART" innerhalb der PL/SQL-Anwendung "PLSQL_APP" gezeigt.

Abbildung 64: URL zum Aufrufen der PL/SQL-Anwendung im Beispiel

```
http://www.oas-server.de/PLSQL_APP/PLSQL_Cart/liste_kunde
```

Nachdem ein Client die Anfrage in Abbildung 64 an den Listener des OAS gesendet hat, leitet der Dispatcher die Anfrage an die PL/SQL-Cartridge weiter. Die Cartridge bearbeitet die Anfrage und führt die PL/SQL-Prozedur in der Datenbank aus. Die Antwort wird in Form eines generierten HTML-Dokumentes an den Client gesendet und dort im Browser angezeigt (siehe Abbildung 65).

Abbildung 65: Anzeige des HTML-Dokumentes im Browser

Liste aller Datensätze der Tabelle Kunde	
Kundenname	Telefon
Albert, Manfred	02202 / 56733
Döpper, Egon	02231 / 34561
Halstenberg, Irene	02432 / 67361
Klasen, Peter	02261 / 33162
Malstenberg, Eva	0221 / 223361
Meier, Karl	0221 / 87762

Einer PL/SQL-Anwendung können beim Aufruf Parameter übergeben werden, indem an die ursprüngliche URL eine Zeichenkette mit Name-Wert-Paaren angehängt wird, wie in Abbildung 66 gezeigt.

Abbildung 66: Parameterübergabe beim Aufruf einer PL/SQL-Prozedur

```
?<paramname>=<paramvalue>&<paramname>=<paramvalue>&...
```

Die mit der URL übergebenen Parameter stehen innerhalb der PL/SQL-Anwendung zur Verfügung.

3.7.2 Perl- und LiveHTML-Anwendungen

Auf dem OAS können mit Perl-Anwendungen und LiveHTML-Anwendungen zwei weitere Anwendungstypen eingesetzt werden, die eng miteinander verbunden sind.

Perl-Anwendung

Perl (Practical Extraction and Report Language) ist eine Interpreter-Programmiersprache, mit der HTML-Dokumente dynamisch erzeugt werden können. Der Quellcode wird mit einem Texteditor erstellt. Die einzelnen Perl-Anweisungen im Quellcode werden zur Laufzeit durch einen Perl-Interpreter interpretiert und ausgeführt.

Beim Einsatz einer Perl-Anwendung auf dem OAS wird eine Perl-Cartridge eingesetzt, die den Perl-Interpreter zur Verfügung stellt. Der OAS 4.0.8.1 unterstützt die Perl Version 5.004_01.

Neben dem Perl-Interpreter stellt der OAS dem Entwickler weitere Zusatzmodule zur Verfügung. So erlaubt ein "Database-Interface" (DBI) den Zugriff auf verschiedene Datenbanken, mittels bestimmter "Database-Driver" (DBD).

Der Einsatz einer Perl-Anwendung auf dem OAS hat gegenüber dem Einsatz auf einem gewöhnlichen Web-Server folgende Vorteile:

- Der Perl-Interpreter wird nicht für jede Anfrage in den Speicher geladen, sondern nur einmalig und bleibt dort aktiv.
- Die Perl-Cartridge kompiliert den Perl-Code einer Anwendung und speichert diesen kompilierten Perl-Code in einem Cache-Bereich. Bei einem Aufruf kann der Perl-Interpreter die Perl-Anwendung sofort ausführen, ohne vorher den Quellcode zu kompilieren.

In Abbildung 67 wird ein Perl-Skript dargestellt, das Kundendaten aus einer Oracle-Datenbank abfragt und auf Basis dieser Daten ein HTML-Dokument generiert.

Abbildung 67: Perl-Skript zur Generierung eines HTML-Dokumentes

```
use DBI; // Zusatzmodul für Datenbankzugriff laden

// Verbindung zu einer Oracle-Datenbank herstellen
// Benutzer=Scott Passwort=tiger Datenbank-Treiber=Oracle
$dbVerbindung = DBI->connect("", "scott", "tiger", "Oracle")
|| die $DBI::errstr;

// Abfrage erstellen
$abfrage = $dbVerbindung->prepare("select kunde_name
                                   from kunde
                                   order by kunde_name")
|| die DBI::errstr;

// Abfrage ausführen
$rc = $abfrage->execute() || die $DBI::errstr;

// HTML-Dokument erzeugen
print "Content-type: text/html\n\n";
print "<html>\n";
print "<body bgcolor=white>\n";
print "<h1>Liste aller Kunden</h1>\n";

// Anzahl der Datensätze ermitteln
$anzahlFelder = $abfrage->rows();

print "Abfrage gibt $anzahlFelder Datensätze zurück<br> \n";

while (($kunde_name) = $abfrage->fetchrow()) {
    print "<br>$kunde_name\n";
}
```

```
print "</body>\n</html>";

// Fehlerbehandlung
warn $DBI::errstr if $DBI::err;
die "fetch error: " . $DBI::errstr if $DBI::err;

$abfrage->finish() || die "Abfrage kann nicht beendet werden";
$dbVerbindung->disconnect() || die "Verbindung nicht beendet";
```

Die Anweisung "use DBI" lädt das Zusatzmodul für den Datenbankzugriff. Hiernach baut die Funktion "connect" eine Verbindung zu einer Oracle-Datenbank unter Verwendung des Oracle-Datenbank-Treibers auf. In der Methode "prepare" wird für diese Verbindung eine Abfrage konstruiert und mit der Methode "execute" ausgeführt. Nach dem Einfügen der Überschrift "<h1>Liste aller Kunden</h1>" in das HTML-Dokument, wird die Anzahl der zurückgegebenen Datensätze mittels der Funktion "rows" abgefragt und in das HTML-Dokument geschrieben. Hiernach werden die abgefragten Kundendaten zeilenweise in das HTML-Dokument eingefügt und das HTML-Dokument mit dem Tag "</HTML>" beendet. Den Schluss des Perl-Skriptes bildet das Schließen der Abfrage und der Datenbankverbindung. Die Anzeige dieses generierten HTML-Dokumentes wird in Abbildung 68 gezeigt.

Abbildung 68: Anzeige des generierten HTML-Dokumentes



Bei Perl-Anwendungen kann der Datenbankzugriff beschleunigt werden, indem während des Initialisierungsprozesses der Cartridge das Zusatzmodul für den Datenbankzugriff und der entsprechende Datenbank-Treiber in den Speicher geladen werden. Somit können

mehrere Anfragen auf das Zusatzmodul und den Datenbank-Treiber direkt zugreifen, ohne diese erneut in den Speicher zu laden.

LiveHTML-Anwendung

LiveHTML ermöglicht die dynamische Erzeugung von HTML-Dokumenten anhand von sogenannten "Template-Files". Template-Files beinhalten statische Abschnitte, die ihren Inhalt nicht verändern, und Anweisungsblöcke, die dynamische Abschnitte zur Laufzeit generieren.

LiveHTML ist vergleichbar mit den von Microsoft entwickelten Active Server Pages (ASP), zur dynamischen Generierung von HTML-Dokumenten. Microsoft setzt in ASP Visual Basic als Programmiersprache für die eingebetteten Anweisungsblöcke ein. Oracle verwendet dagegen in LiveHTML-Anwendungen die Programmiersprache Perl.

In Abbildung 69 wird ein einfaches Template-File zur Erzeugung eines HTML-Dokumentes dargestellt. Durch das Template-File soll eine Textzeile in verschiedenen Schriftgrößen dargestellt werden.

Abbildung 69: Template-File einer LiveHTML-Anwendung

```
<html>
<body>

<% $str = "Live HTML Textzeile";
for($fontsize=3; $fontsize<8; $fontsize++) { %>
  <font size = <% $Response->write($fontsize) %> >
    <p> <% $Response->write($str) %>
  </font>
<% } %>

</body>
</html>
```

Die Tags "<%" und "%>" grenzen die in Perl geschriebenen Anweisungsblöcke ein. Sie stellen die Anwendungslogik dar, mit der die dynamischen Abschnitte des HTML-Dokumentes erzeugt werden. Beim Ausführen der Anwendung werden die Anweisungsblöcke ausgeführt und somit das HTML-Dokument generiert.

Das Schleifenkonstrukt "for(\$fontsize=3; \$fontsize<8; \$fontsize++)" wird zur Wiederholung der Anweisungen "->write(\$str)" verwendet. Diese Anweisung schreibt den auszugebenden String in einen Ausgabestrom.

In Abbildung 70 wird das erzeugte HTML-Dokument und dessen Anzeige im Browser dargestellt.

Abbildung 70: Generiertes HTML-Dokument und dessen Anzeige

<pre> <html> <body> <p> Live HTML Textzeile <p> Live HTML Textzeile <p> Live HTML Textzeile <p> Live HTML Textzeile <p> Live HTML Textzeile </html> </body> </pre>	<p>Live HTML Textzeile</p> <p>Live HTML Textzeile</p> <p>Live HTML Textzeile</p> <p>Live HTML Textzeile</p> <p>Live HTML Textzeile</p>
--	---

Weiterhin wird mit dem OAS für LiveHTML ein IDL-to-Perl-Compiler mit dem Namen "perlidl" mitgeliefert. Mit diesem Compiler ist es möglich für ein CORBA-Objekt eine Schnittstelle in Form eines Perl-Skriptes erzeugen zu lassen. Mit diesem Perl-Skript, das auch Perl-Binding genannt wird, kann auf das entsprechende CORBA-Objekt zugegriffen werden. Somit kann aus LiveHTML-Anwendungen, in denen Perl-Bindings eingesetzt werden, auf CORBA-Objekte zugegriffen werden und diese für die Generierung eines HTML-Dokumentes benutzt werden.

3.7.3 JServlet-Anwendungen

Mit dem OAS 4.0.8.1 können server-seitige Java-Anwendungen eingesetzt werden, die sogenannten Java-Servlets. Dazu enthält die JServlet-Cartridge eine Java Virtual Machine (JVM) und Java-Klassenbibliotheken. Damit steht eine Laufzeitumgebung für server-

seitige Java-Anwendungen zur Verfügung, die nach der Java Servlet API Spezifikation Version 2.1 entwickelt wurden.

Von Vorteil ist, daß die JVM der JServlet-Cartridge nicht für jede Anfrage rauf- und runtergefahren werden muss. Somit können in einer Session Datenbankverbindungen gehalten werden. Daraus resultiert ein Performancegewinn, da nicht für jede Anfrage die Datenbank neu kontaktiert werden muss.

Zur Anwendungsentwicklung für die JServlet-Cartridge wird mit dem OAS das JServlet-Toolkit mitgeliefert, dass aus einer Menge von Java-Klassen besteht. Die Klassen können zur Generierung von HTML-Dokumenten und zum Zugriff auf Datenbanken benutzt werden. Zur Entwicklung der Servlets für die JServlet-Cartridge kann eine Integrierte Entwicklungsumgebung (IDE) eingesetzt werden, wie z. B. Oracle JDeveloper.

Grundlagen zu Java-Servlets

Ein Java-Servlet besteht in seinem grundsätzlichen Aufbau immer aus mindestens drei Methoden:

- (1) `init()`
- (2) `doGet()` / `doPost()`
- (3) `destroy()`

(1) `init()`

Die Methode `init()` wird während der Initialisierung eines Servlets implizit aufgerufen. Zu beachten ist, dass `init()` für jedes Servlet genau einmal aufgerufen wird. Tritt während der Ausführung von `init()` eine Exception auf, so ist die JServlet-Cartridge-Instanz nicht verfügbar. Die JServlet-Cartridge gibt dann das Scheitern der Anfrage in Form einer Fehlermeldung an den Client zurück.

(2) doGet() / doPost()

Diese Methoden sind Einstiegsmethoden in ein HTTP-Servlet. Die Methode "doGet" wird bei der HTTP-Methode "GET" verwendet und "doPost" bei der HTTP-Methode "POST". Die HTTP-Methoden GET und POST sind im HTTP definiert und spezifizieren die Übergabe von HTML-Formularen zwischen Client und Server. Es kann eine oder es können beide Methoden als Einstiegspunkte in ein HTTP-Servlet implementiert werden.

(3) destroy()

Beim Beenden einer JServlet-Anwendung wird implizit dessen Methode destroy() aufgerufen. Diese Methode wird für jedes Servlet genau einmal aufgerufen, allerdings erst nachdem alle Cartridge-Instanzen ihre Antworten abgeschlossen haben, oder ihr Time-Out erreicht haben. In dieser Methode werden meist Systemressourcen wieder freigegeben.

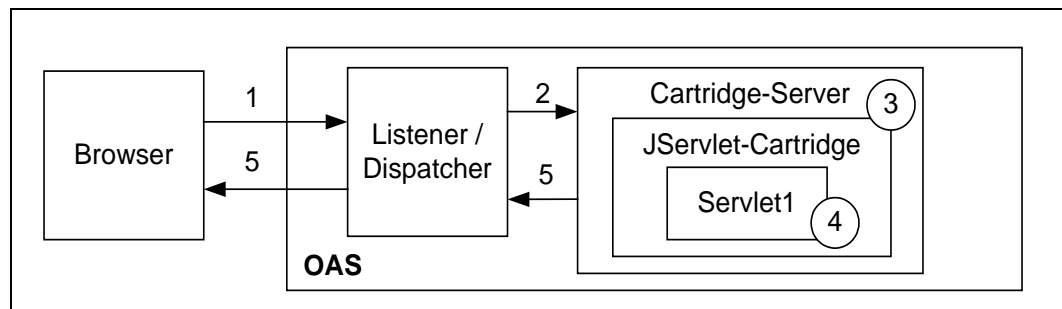
Ablauf einer Anfrage an eine JServlet-Cartridge

In Abbildung 71 wird dargestellt, wie eine Anfrage an eine JServlet-Cartridge vom OAS bearbeitet wird.

Ablauf der Bearbeitung:

1. Der Listener empfängt die Anfrage eines Client an eine JServlet-Cartridge. Eine Anfrage könnte sein: `http://pc60.opitz.de/servlets/Servlet1`.
2. Der Dispatcher untersucht die URL der Anfrage und stellt fest, dass die Anfrage an eine JServlet-Cartridge gerichtet ist. Er erkennt dies daran, dass der virtuelle Pfad (`/servlets`) einer JServlet-Cartridge zugeordnet ist. Die Anfrage wird an die entsprechende JServlet-Cartridge übergeben.
3. Die JServlet-Cartridge, die innerhalb eines Cartridge-Server ausgeführt wird, stellt aufgrund der URL fest, dass die Java-Klasse "Servlet1" aufgerufen werden soll. Der Name der Servlet-Klasse steht immer am Ende der URL.

Abbildung 71: Anfrage an eine JServlet-Cartridge



Quelle: In Anlehnung an: Oracle Application Server Documentation, Release 4.0.8.1: Developer's Guide: JServlet Applications, S. 1-4.

4. Die JServlet-Cartridge führt die angeforderte Java-Klasse aus, indem sie deren Einstiegsmethode aufruft.
5. Die Java-Klasse generiert eine Antwort (HTML-Dokument) und gibt sie über einen speziellen Ausgabestrom zurück an die JServlet-Cartridge. Danach wird das generierte HTML-Dokument in umgekehrter Richtung an den Client-Browser zurückgegeben.

Beispiel

In diesem Abschnitt soll gezeigt werden, wie ein Java-Servlet zu erstellen ist, der OAS für dessen Einsatz konfiguriert wird und das Servlet aufzurufen ist. Um eine JServlet-Anwendung zu erstellen, bedarf es der folgenden Schritte:

- (1) Erstellen der Java-Klassendatei
- (2) Anlegen einer JServlet-Anwendung und einer JServlet-Cartridge
- (3) Aufrufen der JServlet-Cartridge

(1) Erstellen der Java-Klassendatei

Der Quellcode des Servlets wird mit einem normalen Editor oder einer integrierten Entwicklungsumgebung erstellt und unter dem Namen "Servlet1.java" gespeichert. In Abbildung 72 wird der Quellcode eines HTTP-Servlet gezeigt.

Abbildung 72: Quellcode eines HTTP-Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Servlet1 extends HttpServlet {
    // doGet-Methode als Einstiegsmethode
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {
        //MIME-Typ setzen
        response.setContentType("text/html");

        //Writer für Ausgabestrom initialisieren
        PrintWriter out = response.getWriter();

        //HTML-Dokument generieren
        out.println("<html><body>");
        out.println("<h2>Hello World</h2>");
        out.println("</body></html>");
    }
}
```

Dieses Java-Servlet generiert ein HTML-Dokument, das den Schriftzug "Hello World" darstellt. Der Quellcode wird mit einem Java-Bytecode-Compiler in eine Klassendatei mit dem Namen "Servlet1.class" kompiliert. Danach wird diese Datei in ein Verzeichnis auf dem OAS kopiert. Dieses Verzeichnis wird beim Konfigurieren der Anwendung einer JServlet-Cartridge über einen virtuellen Pfad zugeordnet¹.

(2) Anlegen einer JServlet-Anwendung und der JServlet-Cartridge

Die JServlet-Anwendung und die JServlet-Cartridge werden mit Hilfe des OAS-Manager angelegt und konfiguriert. Folgende Schritte sind dazu durchzuführen:

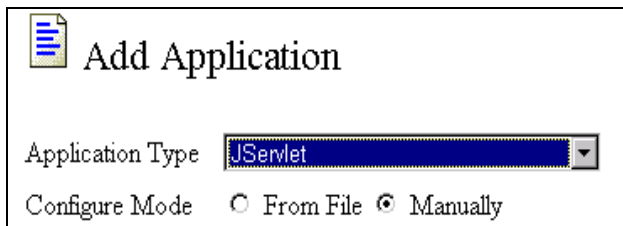
- (a) Anwendungstyp auswählen
- (b) Name der Anwendung angeben
- (c) Cartridge zur Anwendung hinzufügen und Parameter eintragen
- (d) Reload des OAS

¹ Siehe Abbildung 75.

(a) Anwendungstyp auswählen

In diesem Schritt wird der Anwendungstyp JServlet aus der Liste "Application Type" ausgewählt (siehe Abbildung 73).

Abbildung 73: Anwendungstyp JServlet auswählen



Add Application

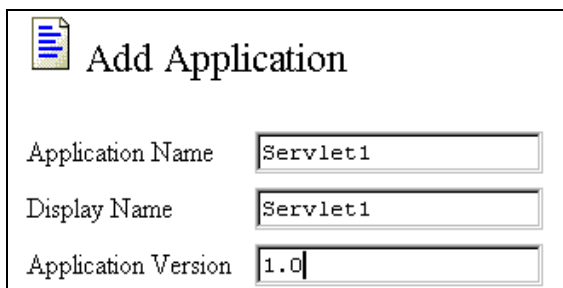
Application Type: JServlet

Configure Mode: ☐ From File ☒ Manually

(b) Name der Anwendung angeben

In diesem Schritt wird der Name der Anwendung, wie er intern verwendet wird (Application Name), wie er angezeigt wird (Display Name) und eine Versionsnummer angegeben (siehe Abbildung 74).

Abbildung 74: Anwendungsnamen angeben



Add Application

Application Name: Servlet1

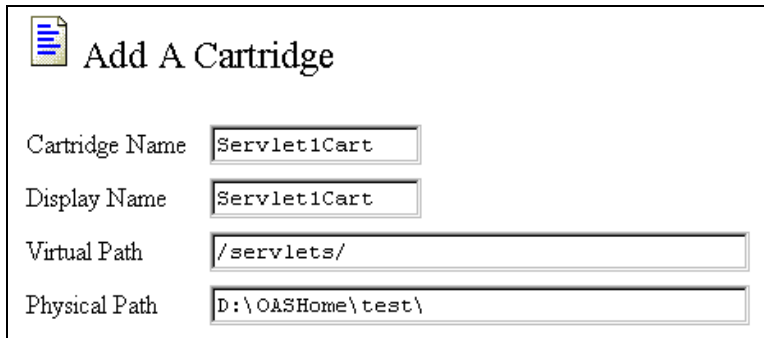
Display Name: Servlet1

Application Version: 1.0

(c) Cartridge zur Anwendung hinzufügen und Parameter eintragen

In diesem Schritt wird der Name der Cartridge, wie er intern verwendet (Cartridge Name) und wie er angezeigt wird (Display Name) angegeben. Weiterhin ist der virtuelle Pfad und der physikalische Pfad anzugeben, in dem die Klassendateien des Servlet zu finden sind (siehe Abbildung 75).

Abbildung 75: Cartridge hinzufügen



Cartridge Name	Servlet1Cart
Display Name	Servlet1Cart
Virtual Path	/servlets/
Physical Path	D:\OASHome\test\

(d) Reload des OAS

Als letzter Schritt zum Installieren einer JServlet-Anwendung, muss ein Reload des OAS durchgeführt werden.

(3) Aufrufen der JServlet-Cartridge

Die JServlet-Cartridge kann aus einem Browser heraus aufgerufen werden, indem folgende URL eingegeben wird: `http://<OASHost>/servlets/Servlet1`. Es besteht ebenfalls die Möglichkeit aus einem HTML-Dokument heraus die JServlet-Cartridge aufzurufen. Dazu wird in diesem HTML-Dokument ein Link mit der oben stehenden URL definiert.

Über JDBC¹ kann ein Java-Servlet auf eine Datenbank zugreifen, wenn für die entsprechende Datenbank ein JDBC-Treiber zur Verfügung steht.

3.7.4 C++-CORBA-Anwendungen

Nachdem im Kapitel "2.4 CORBA" die Grundlagen von CORBA beschrieben wurden, wird hier konkret auf in C++ geschriebene CORBA-Anwendungen eingegangen, wie sie auf dem OAS eingesetzt werden.

Der OAS unterstützt komponentenbasierte Anwendungen in Form von C++-CORBA-Anwendungen. Eine C++-CORBA-Anwendung auf dem OAS wird in der

¹ Siehe Kapitel "3.4 Datenbankzugriff".

Programmiersprache C++ implementiert und enthält eine oder mehrere C++-CORBA-Cartridges.

Im Folgenden wird eine C++-CORBA-Anwendung als C++-Anwendung und eine C++-CORBA-Cartridge als C++-Cartridge bezeichnet.

Eine C++-Cartridge enthält die Anwendungslogik. Um die C++-Cartridge ausführen zu können, wird vom OAS ein C++-Cartridge-Container als Laufzeitumgebung bereitgestellt. Der C++-Cartridge-Container erstellt und verwaltet dabei die einzelnen C++-Cartridge-Instanzen und stellt ihnen weitere Dienste zur Verfügung. So übernimmt der C++-Cartridge-Container die Transaktionsverwaltung, das Sicherheitsmanagement, sowie weitere Dienste und entlastet den Entwickler davon, diese Dienste selbst implementieren zu müssen. Der C++-Cartridge-Container wird in einem Cartridge-Server auf dem OAS ausgeführt.

In C++-CORBA-Anwendungen, die auf dem OAS eingesetzt werden, sind zwei Objektbegriffe zu unterscheiden:

- (1) C++-Objekt
- (2) C++-Implementierungsobjekt

(1) C++-Objekt

Ein C++-Objekt ist eine Instanz einer C++-Cartridge. Jedes C++-Objekt hat eine Objektreferenz. Ein Client benutzt diese Objektreferenz, um eine Methode des C++-Objektes aufzurufen. Ein C++-Objekt lebt vom Zeitpunkt seiner Erzeugung bis zum Zeitpunkt seiner Löschung innerhalb eines C++-Cartridge-Containers.

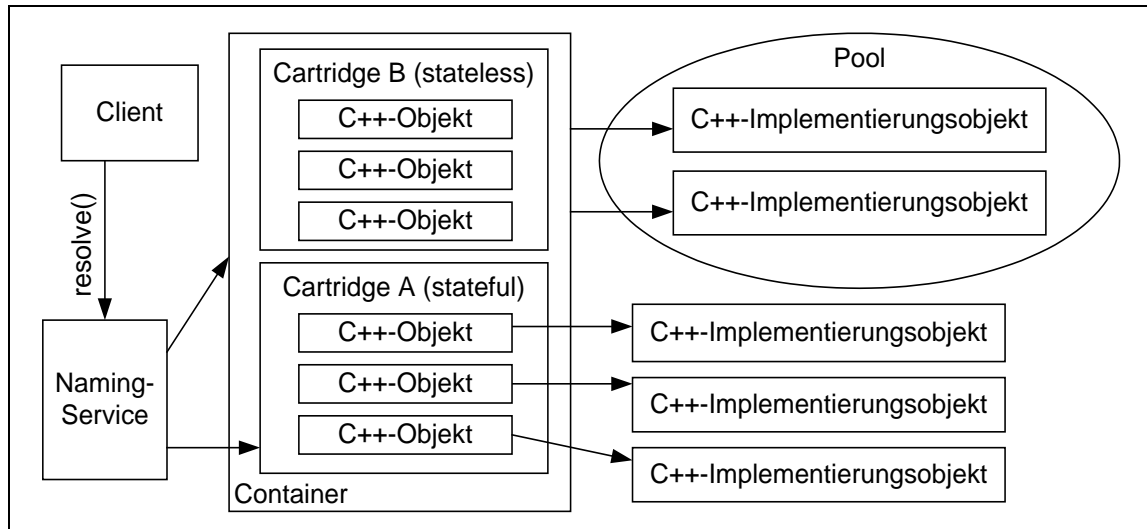
(2) C++-Implementierungsobjekt

Ein C++-Implementierungsobjekt ist eine Instanz der vom Entwickler geschriebenen C++-Implementierungsklasse.

Stateless and stateful C++-Cartridge

In einer C++-Anwendung können die C++-Cartridges entweder als "stateless" oder "stateful" definiert werden. In Abbildung 76 werden diese beiden Möglichkeiten dargestellt.

Abbildung 76: Stateful und stateless C++-Cartridge



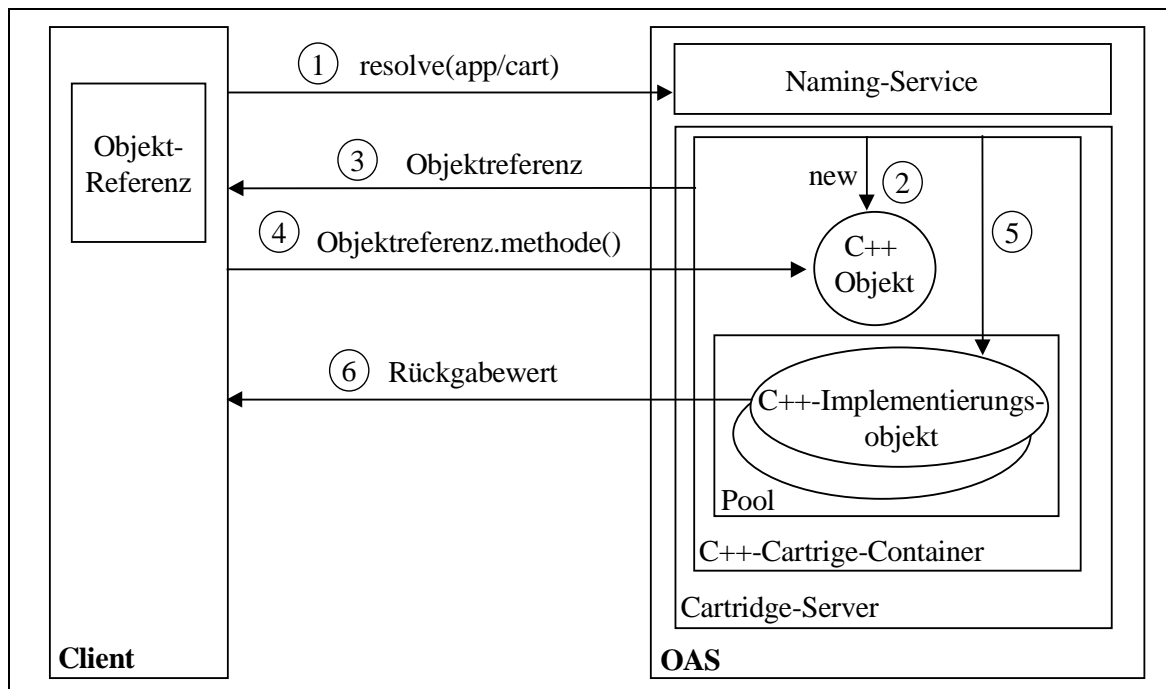
Quelle: Oracle Application Server Documentation, Release 4.0.8.1: Developer's Guide: C++ CORBA Applications, S. 3-10.

Eine stateless C++-Cartridge (Cartridge B) erlaubt keinen zustandsbehafteten Dialog zwischen Client und C++-Implementierungsobjekt.

Der C++-Cartridge-Container erzeugt einen Pool von C++-Implementierungsobjekten. Bei jedem Aufruf der Methode "resolve" wird ein C++-Objekt erzeugt. Der Container ordnet für jeden Methodenaufruf dem entsprechenden Client ein existierendes oder ein neues C++-Implementierungsobjekt aus dem Pool zu. Ein C++-Implementierungsobjekt ist somit nicht für seine Lebensdauer an einen Client gebunden. Durch den Pool von C++-Implementierungsobjekten kann eine große Anzahl von Clients eine geringe Anzahl von C++-Implementierungsobjekten nutzen.

Um eine stateless C++-Cartridge von einem entfernten Client aufrufen zu können, müssen die in Abbildung 77 dargestellten Schritte durchgeführt werden.

Abbildung 77: Aufruf einer stateless C++-Cartridge



Quelle: In Anlehnung an: Oracle Application Server Documentation, Release 4.0.8.1:
Developer's Guide: C++ CORBA Applications, S. 3-13.

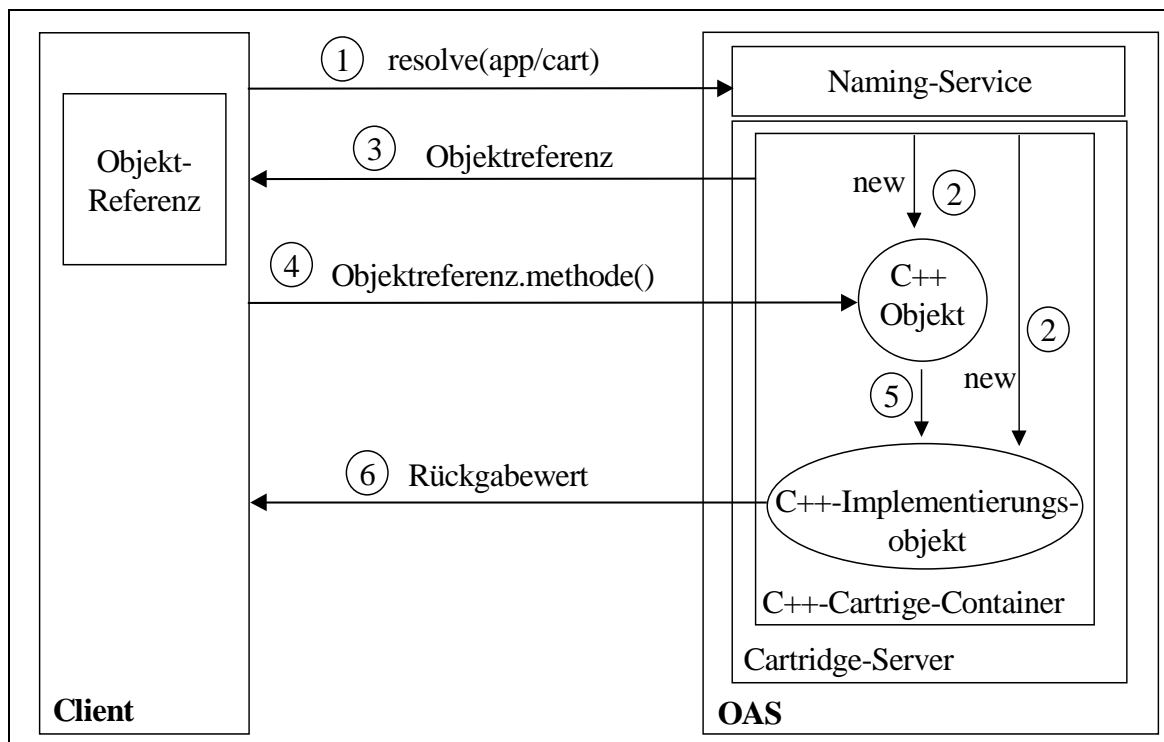
1. Alle Namen der C++-Objekte, die auf dem OAS installiert sind, verwaltet der OAS in einem Namensverzeichnis. Dieses Namensverzeichnis wird über den "Naming-Service" verwaltet und abgefragt. Um eine Objektreferenz auf ein C++-Objekt zu erhalten, sendet der Client die Anfrage "resolve(<app>/<cart>)" an den Naming-Service.
2. Findet der Naming-Service im Namensverzeichnis den Namen des angeforderten C++-Objektes, erzeugt der OAS einen Cartridge-Server. Der C++-Cartridge-Container instanziiert das C++-Objekt.
3. Der C++-Cartridge-Container sendet die Objektreferenz des C++-Objektes zurück an den Client.
4. Der Client ruft über die Objektreferenz die Methoden des C++-Objektes auf.
5. Der C++-Cartridge-Container delegiert den Methodenaufruf an ein freies C++-Implementierungsobjekt im Pool. Sollte kein freies C++-Implementierungsobjekt existieren, wird vom C++-Cartridge-Container ein neues erzeugt und der Methodenaufruf zu diesem delegiert.

6. Der Rückgabewert der Methode wird vom C++-Implementierungsobjekt an den Client zurückgegeben.

Bei einer stateful C++-Cartridge (Cartridge A in Abbildung 76) wird bei jedem Aufruf der Methode "resolve" ein C++-Objekt und ein C++-Implementierungsobjekt erzeugt. Das C++-Implementierungsobjekt ist von seiner Erzeugung bis zu seiner Löschung an einen Client gebunden. Eine stateful Cartridge erlaubt somit eine zustandsbehaftete Interaktion zwischen Client und C++-Implementierungsobjekt.

Um eine stateful C++-Cartridge von einem entfernten Client aufrufen zu können, müssen die in Abbildung 78 dargestellten Schritte durchgeführt werden.

Abbildung 78: Aufruf einer stateful C++-Cartridge



Quelle: In Anlehnung an: Oracle Application Server Documentation, Release 4.0.8.1:
Developer's Guide: C++ CORBA Applications, S. 3-11.

1. Um eine Objektreferenz auf ein C++-Objekt zu erhalten, sendet der Client die Anfrage "resolve(<app>/<cart>)" an den Naming-Service.

2. Findet der Naming-Service im Namensverzeichnis den Namen des angeforderten C++-Objektes, erzeugt der OAS einen Cartridge-Server. Der C++-Cartridge-Container instanziiert das C++-Objekt und das C++-Implementierungsobjekt.
3. Der C++-Cartridge-Container sendet die Objektreferenz des C++-Objektes zurück an den Client.
4. Der Client ruft über die Objektreferenz die Methoden des C++-Objektes auf.
5. Das C++-Objekt delegiert die Anfrage an das entsprechende C++-Implementierungsobjekt.
6. Der Rückgabewert der Methode wird vom C++-Implementierungsobjekt an den Client zurückgegeben.

Ob eine C++-Cartridge stateful oder stateless ist, wird im Deployment-Descriptor der entsprechenden C++-Anwendung festgelegt. Defaultmäßig wird eine C++-Cartridge als stateful installiert.

Ein Deployment-Descriptor wird für jede auf dem OAS installierte C++-Anwendung benötigt. Er beinhaltet den Namen der C++-Anwendung, eine Auflistung der enthaltenen C++-Cartridges und weitere Einstellungen für die C++-Cartridges. Die Einstellungen im Deployment-Descriptor werden bei der Installation als Standardeinstellungen vom OAS übernommen und können über den OAS-Manager geändert werden.

Beispiel

Im Folgenden werden die Schritte erläutert, die notwendig sind, um eine C++-Anwendung zu entwickeln und diese auf dem OAS zu installieren.

Am Beispiel eines Bankkontos, das als C++-Objekt mit dem Namen "Konto" implementiert wird, sollen diese Schritte erläutert werden.

Das Bankkonto hat einen bestimmten Kontostand, der mit der Methode "getKontostand" abgefragt werden kann. Zudem sollen über die Methode "einzahlen" bestimmte Beträge auf das Bankkonto eingezahlt werden können.

Zur Realisierung dieses Beispiels müssen die folgenden fünf Schritte durchgeführt werden:

- (1) Erstellen des IDL-File
- (2) Erstellen der C++-Implementierungsklasse
- (3) Erstellen des Deployment-Descriptor-File
- (4) Installieren der Anwendung auf dem OAS
- (5) Erstellen des Client

(1) Erstellen des IDL-File

Im ersten Schritt wird ein IDL-File erstellt, das die Schnittstelle des C++-Objektes widerspiegelt. In diesem IDL-File werden die Methoden deklariert, die das C++-Objekt den Clients zur Verfügung stellt.

Abbildung 79: IDL-File einer C++-Anwendung

```
// IDL-File "Bank.idl"
//Bank-Anwendung
module Bank
{
    interface Konto
    {
        short getKontostand();
        void einzahlen(in short betrag);
    }
}
```

In Abbildung 79 ist ein IDL-File dargestellt, welches für die C++-Anwendung mit dem Namen "Bank" die Schnittstelle des C++-Objektes definiert. In diesem Beispiel besitzt die C++-Anwendung lediglich ein C++-Objekt mit dem Namen "Konto", welches die beiden Methoden "getKontostand" und "einzahlen" hat.

(2) Erstellen der C++-Implementierungsklasse

Im zweiten Schritt wird für jedes C++-Objekt die Anwendungslogik in einer C++-Implementierungsklasse implementiert.

Für die Erstellung einer C++-Implementierungsklasse sind zwei Files notwendig. Ein C++-Header-File, das die Methoden und Variablen der C++-Implementierungsklasse deklariert und ein C++-Body-File, in dem die einzelnen Methoden implementiert werden. In Abbildung 80 wird ein Codefragment eines C++-Header-File mit dem Namen "KontoImp.h" gezeigt.

Abbildung 80: Codefragment eines C++-Header-File

```
//Deklaration des C++-Objektes Bank::Konto
//Dateiname: "KontoImp.h"

class KontoImpl : public oas::cpp::Object
{
private:
    short m_kontostand;

public:
    KontoImpl() {}

    short getKontostand();
    void einzahlen(short betrag);
};
```

In Abbildung 81 ist ein Codefragment des benötigten C++-Body-File mit dem Namen "KontoImp.cpp" dargestellt. In ihm werden die einzelnen Methoden implementiert, die im C++-Header-File deklariert sind.

Abbildung 81: Codefragment eines C++-Body-File

```
// aktuellen Kontostand zurückgeben
CORBA::Short KontoImpl::getKontostand()
{
    return m_Kontostand;
}

// Bank::Konto::einzahlen Implementierung
void KontoImpl::einzahlen(CORBA::Short Betrag)
{
    // Kontostand aktualisieren
    m_kontostand = m_kontostand + betrag;
}
```

(3) Erstellen des Deployment-Descriptor-File

Für jede C++-Anwendung, die auf dem OAS installiert werden soll, muss ein Deployment-Descriptor-File erstellt werden. In Abbildung 82 ist ein Deployment-Descriptor-File für die C++-Anwendung "Bank" gezeigt.

Abbildung 82: Deployment-Descriptor-File einer C++-Anwendung

```
Name = Bank

[Konto]
stateless = false
remoteInterface = Bank::Konto
implementationClass = KontoImpl
implementationHeader = KontoImpl.h
```

In diesem Deployment-Descriptor-File wird definiert, dass die Anwendung "Bank" ein C++-Objekt mit dem Namen "Konto" enthält. Das C++-Objekt wird in einer stateless C++-Cartridge ausgeführt.

(4) Installieren der Anwendung auf dem OAS

Um die Anwendung einsetzen zu können, müssen vier weitere Schritte durchgeführt werden, die an dieser Stelle beschrieben werden:

- a) Auf Grundlage des IDL-File "Bank.idl" müssen mit Hilfe eines IDL-C++ Compilers ("oasoidlc") Stubs¹ und Skeletons² erzeugt werden.
- b) Mit dem Dienstprogramm "cppgen" des OAS muss eine C++-Cartridge-Factory erstellt werden. Die C++-Cartridge-Factory erstellt bei Bedarf eine neue C++-Cartridge-Instanz.
- c) Es muss ein Library-File mit Hilfe eines C++-Compilers erstellt werden, das die Files enthält, die von der C++-Anwendung benötigt werden. Die Erstellung ist abhängig vom Betriebssystem.

¹ Siehe Kapitel "2.4 CORBA".

² Siehe Kapitel "2.4 CORBA".

- d) Mit dem Dienstprogramm "cppinstaller" des OAS wird die Anwendung auf dem OAS installiert.

(5) Erstellen des Client

In Abbildung 83 wird das Codefragment eines Client dargestellt, der die Methoden des entfernten C++-Objektes "Konto" aufruft.

Um von einem Client auf die Methoden eines entfernten C++-Objektes zugreifen zu können, muss über den Naming-Service eine Objektreferenz auf das C++-Objekt angefordert werden. In Abbildung 83 wird hierfür mit Übergabe des Anwendungsnamens "Bank" und des Cartridgenamens "Konto" die Referenz des C++-Objektes beim Naming-Service angefordert.

Abbildung 83: Codefragment des Client eines C++-Objekt

```
CosNaming::Name name;
name.length(2);
name[0].id = CORBA::string_dup("Bank");//Name der C++-Anwendung
name[1].id = CORBA::string_dup("Konto");//Name der C++-Cartridge

try {

    // Objektreferenz für das C++-Objekt beim Naming-Service des OAS
    // anfordern
    CORBA::Object_ptr cppobj_ptr = rootNC_var->resolve(name);

    // Typumwandlung der erhaltenen Objektreferenz
    Bank::Konto_ptr konto_ptr = Bank::Konto::_narrow(cppobj_ptr);
    Bank::Konto_var konto_var = konto_ptr;

    // Aufrufen von Methoden
    cout << "Kontostand:" << konto_var->getKontostand() << endl;
    konto_var->einzahlen(100);
    cout << "Einzahlen: 100 DM." << endl;
    cout << "Kontostand:" << konto_var->getKontostand() << endl;
}

catch ...
```

Die erhaltene Objektreferenz wird durch die Methode "_narrow" in den Typ des C++-Objektes umgewandelt. Mit Hilfe dieser Objektreferenz können die Methodenaufrufe durchgeführt werden. So kann mit der Methode "getKontostand" der aktuelle

Kontostand abgefragt werden und mit der Methode "einzahlen" eine Einzahlung getätigt werden.

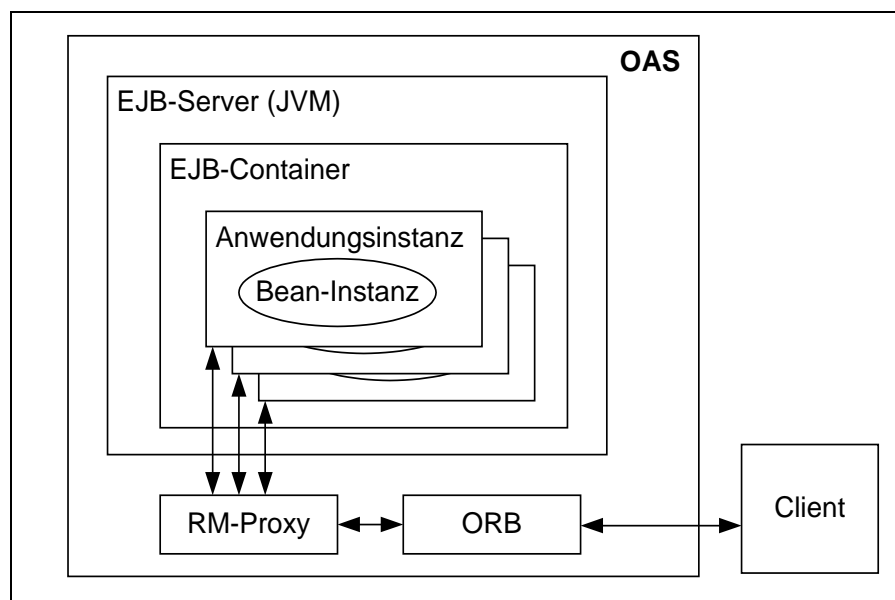
3.7.5 EJB-Anwendungen

Nachdem im Kapitel "2.5 Enterprise JavaBeans" eine Einführung in das Thema Enterprise JavaBeans (EJB) gegeben wurde, wird hier konkret auf EJB-Anwendungen eingegangen, wie sie auf dem OAS eingesetzt werden.

Auf dem OAS ist die EJB-Server- und EJB-Container-Infrastruktur in einer CORBA-Umgebung implementiert. Das bedeutet Enterprise JavaBeans werden wie CORBA-Objekte behandelt, die mit anderen CORBA-Objekten kommunizieren können.

Zunächst wird die Architektur der auf dem OAS eingesetzten EJB-Anwendungen beschrieben. In Abbildung 84 wird diese Architektur gezeigt.

Abbildung 84: EJB-Architektur auf dem OAS



Quelle: Oracle Application Server Documentation, Release 4.0.8.1: Developer's Guide: EJB, ECO/Java and CORBA Applications, S. 2-3.

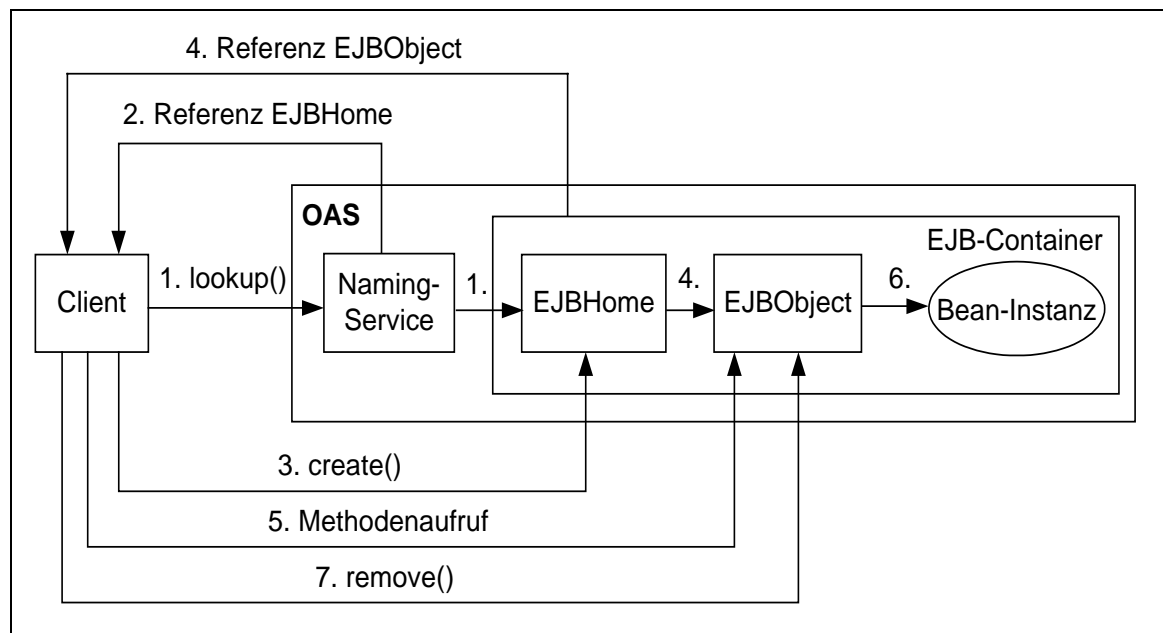
In einer EJB-Anwendung existiert eine Bean-Instanz in einer Anwendungsinstanz. Die Anwendungsinstanz wird in einem EJB-Container ausgeführt. Der EJB-Container stellt verschiedene Dienste zur Verfügung, wie Lebenszyklus-Management, Transaktionsverwaltung und Sicherheitsmanagement. Ein EJB-Container wird innerhalb eines EJB-Server ausgeführt. Ein EJB-Server ist ein Prozess auf dem OAS und bietet eine JVM und Java-Packages, die zur Ausführung der Beans benötigt werden.

Der RM-Proxy des OAS bietet mit Hilfe des Auth-Server eine Authentifizierung der Clients, die sich mit der Anwendungsinstanz verbinden. Nachdem ein Client einer Anwendungsinstanz zugeordnet worden ist, läuft die Kommunikation zwischen Client und Anwendungsinstanz ausschließlich über den EJB-Server.

Zugriff eines Client auf eine Session-Bean

In Abbildung 85 werden die Schritte dargestellt, die für den Zugriff eines Client auf eine Session-Bean notwendig sind.

Abbildung 85: Zugriff eines Client auf eine Session-Bean



1. Der Client fordert mit der Methode "lookup" die Referenz auf das EJBHome beim Naming-Service an. Daraufhin erzeugt der Container das EJBHome, falls es noch nicht existiert.
2. Die Referenz auf das EJBHome wird vom Naming-Service an den Client zurückgegeben.
3. Der Client ruft die Methode "create" des EJBHome auf. Der Container erzeugt eine Bean-Instanz und ein EJBObject.
4. Die Referenz auf das EJBObject wird vom Container an den Client zurückgegeben.
5. Der Client benutzt die Referenz auf das EJBObject um Methoden der Bean-Instanz aufzurufen.
6. Der Container delegiert den Methodenaufruf vom EJBObject an die Bean-Instanz.
7. Wenn die Bean-Instanz nicht mehr benötigt wird, ruft der Client die Methode "remove" des EJBObject auf. Dadurch wird das EJBObject und die Bean-Instanz gelöscht, sofern es sich bei der Bean um eine stateful Session-Bean handelt.

Entwicklung und Installation einer EJB-Anwendung auf dem OAS

Die Entwicklung und Installation einer EJB-Anwendung auf dem OAS wird im Folgenden anhand eines Beispiels beschrieben.

In diesem Beispiel soll eine Session-Bean erzeugt werden, die eine Stack-Logik (Stapelverarbeitung) implementiert. Auf diesen Stack können neue Elemente mit der Methode "push" gelegt werden. Vorhandene Elemente können mit der Methode "pop" vom Stack entfernt werden. Außerdem kann die maximale Anzahl der Elemente des Stack mit der Methode "setStackSize" festgelegt werden.

Für die Entwicklung und Installation der EJB-Anwendung auf dem OAS sind folgende Schritte notwendig:

1. Erstellen der Bean-Klasse
2. Erstellen von Home-Interface und Remote-Interface
3. Erstellen der Hilfsklassen
4. Kompilieren der Klassen

5. Erstellen von Deployment-Descriptors und Manifest-File
6. Erstellen des ejb-jar-File
7. Einrichten der Anwendung auf dem OAS
8. Erstellen des Client
9. Client ausführen

1. Erstellen der Bean-Klasse

Im ersten Schritt wird die Bean-Klasse erzeugt. In Abbildung 86 wird der Quellcode einer Session-Bean-Klasse gezeigt, die eine Stack-Logik (Stapelverarbeitung) implementiert.

Abbildung 86: Quellcode einer Session-Bean-Klasse

```
import javax.ejb.*;
import oracle.oas.ejb.*;
import javax.naming.*;

public class ServerStack implements javax.ejb.SessionBean {

    private int stackSize = 0;
    private String stackElements[];
    private int top = -1;

    // Methode, die beim Erstellen einer Bean-Instanz aufgerufen
    // wird
    public void ejbCreate() throws CreateException {
        setStackSize(10);
    }

    // Methode, die beim Löschen einer Bean-Instanz aufgerufen
    // wird
    public void ejbRemove() throws RemoveException {}

    // Methode gibt die Stack-Größe zurück
    public int getStackSize() {
        return stackSize;
    }

    // Methode setzt die Stack-Größe
    public void setStackSize(int size) throws StackException {
        if (size < 0)
            throw new StackException();

        logger.println("Setze Stack-Größe auf " + size);

        top = -1;
        stackSize = size;
    }
}
```

```
    stackElements = new String[size];
}

// Methode, die ein Element auf den Stack legt
public void push(String value) throws StackException {
    if (top == stackSize - 1)
        throw new StackException();
    stackElements[++top] = value;
}

// Methode, die ein Element vom Stack herunternimmt
public String pop() throws StackException {
    if (top == -1)
        throw new StackException();
    return stackElements[top--];
}
}
```

In der Bean-Klasse werden die Methoden "getStackSize", "setStackSize", "push" und "pop" implementiert. Deren Signaturen sind im Remote-Interface definiert¹. Die Methode "ejbCreate" wird beim Erzeugen jeder neuen Bean-Instanz aufgerufen und die Methode "ejbRemove" wird beim Löschen jeder Bean-Instanz aufgerufen.

Da es sich um eine Session-Bean handelt, muss die Bean-Klasse das Interface "javax.ejb.SessionBean" implementieren.

2. Erstellen von Home-Interface und Remote-Interface

Zum Erzeugen und Löschen von Bean-Instanzen muss ein Home-Interface erstellt werden. In Abbildung 87 wird der Quellcode eines Home-Interface dargestellt.

Abbildung 87: Quellcode eines Home-Interface

```
import javax.ejb.*;
import java.rmi.RemoteException;

public interface ServerStackHome extends EJBHome {
    public ServerStackRemote create() throws CreateException,
                                         RemoteException;
}
```

¹ Siehe Abbildung 88.

Das Home-Interface stellt dem Client die Methode "create" zur Verfügung, mit der er neue Bean-Instanzen erzeugen kann. Ein Home-Interface erweitert immer das Interface "javax.ejb.EJBHome".

Um auf die Methoden einer Bean zugreifen zu können, muss ein Remote-Interface erstellt werden. Im Remote-Interface werden die Signaturen aller Methoden definiert, die ein Client von einer Bean-Instanz aufrufen kann. Implementiert werden diese Methoden in der Bean-Klasse. In Abbildung 88 wird der Quellcode eines Remote-Interface gezeigt. Ein Remote-Interface erweitert immer das Interface "javax.ejb.EJBObject".

Abbildung 88: Quellcode eines Remote-Interface

```
public interface ServerStackRemote extends javax.ejb.EJBObject {  
    public int getStackSize()  
        throws java.rmi.RemoteException;  
  
    public void setStackSize(int size)  
        throws StackException,  
            java.rmi.RemoteException;  
  
    public void push(String value)  
        throws StackException,  
            java.rmi.RemoteException;  
  
    public String pop()  
        throws StackException,  
            java.rmi.RemoteException;  
}
```

Da das Home-Interface und das Remote-Interface vom Client verwendet werden und die einzelnen Methodenaufrufe über das Netzwerk zum Server übertragen werden, kann jede Methode eine "RemoteException" auslösen. Diese Exceptions signalisieren, dass bei der Übertragung über das Netzwerk ein Fehler aufgetreten ist.

3. Erstellen der Hilfsklassen

Zusätzlich zu Bean-Klasse, Home-Interface und Remote-Interface können weitere Hilfsklassen implementiert werden, z. B. Hilfsklassen zur Behandlung von Ausnahmen.

4. Kompilieren der Klassen

Nachdem die einzelnen Klassen und Schnittstellen erstellt wurden, müssen diese mit einem Java-Bytecode-Compiler kompiliert werden.

5. Erstellen von Deployment-Descriptors und Manifest-File

Für eine EJB-Anwendung auf dem OAS können mehrere Deployment-Descriptors erstellt werden:

- Deployment-Descriptor für die Bean
- Deployment-Descriptor für die Anwendung (optional)

Die Deployment-Descriptors werden durch eine Java-Anwendung erstellt. In Abbildung 89 wird der Quellcode zum Erstellen der Deployment-Descriptors dargestellt.

Abbildung 89: Quellcode zum Erstellen der Deployment-Descriptors

```
import javax.ejb.deployment.*;
import oracle.oas.ejb.deployment.*;
import javax.naming.CompositeName;
import java.util.Properties;
import java.io.*;
import java.lang.reflect.*;

public class ServerStackDescriptor
{
    public static void main(String[] args)
    {
        FileOutputStream fos;
        ObjectOutputStream oos;

        OASApplicationDescriptor ad = new OASApplicationDescriptor();
        SessionDescriptor sd = new SessionDescriptor();
        ControlDescriptor cd = new ControlDescriptor();
        ControlDescriptor cdAll[] = new ControlDescriptor[1];

        try
        {
            //-----
            // Erzeugen des Deployment-Descriptor für die Anwendung
            //-----
            // Name der Anwendung festlegen
```

```

        ad.setBeanHomeName (new CompositeName("StackDemo"));
        ad.setTxEnabled(false); // Transaktion deaktivieren

        // Erzeugen der Deployment-Descriptor File
        fos = new FileOutputStream("StackDemoDeployment.ser");
        oos = new ObjectOutputStream(fos);
        oos.writeObject(ad); // Schreiben in die Datei
        oos.flush();

        //-----
        // Erzeugen des Deployment-Descriptor für die Bean
        //-----
        // Name der Bean festlegen
        sd.setEnterpriseBeanClassName("myStack.ServerStack");
        //Bean Parameter setzen
        Properties p = new Properties();
        p.put("initialStackSize", "20");
        sd.setEnvironmentProperties(p);

        //Angabe der Namen für Bean-Klasse, Remote- und Home-
        //Interface
        sd.setBeanHomeName(new CompositeName("ServerStack"));
        sd.setRemoteInterfaceClassName("myStack.ServerStackRemote");
        sd.setHomeInterfaceClassName("myStack.ServerStackHome");

        // Bean als zustandsbehaftet(stateful) definieren
        sd.setStateManagementType(sd.STATEFUL_SESSION);

        // Erzeugen des Deployment-Descriptor File
        fos = new FileOutputStream("ServerStackDeployment.ser");
        oos = new ObjectOutputStream(fos);
        oos.writeObject(sd); // Schreiben in die Datei
        oos.flush();
    }
    catch ...
}
}

```

Der Quellcode erzeugt die Deployment-Descriptors in Form von zwei Deployment-Descriptor-Files. "ServerStackDeployment.ser" ist der Deployment-Descriptor für die Session-Bean und "StackDemoDeployment.ser" ist der Deployment-Descriptor für die Anwendung.

Neben den Deployment-Descriptors wird ein Initial Manifest-File benötigt, welches das in Schritt 6 erzeugte jar-File als ejb-jar-File identifiziert und die Namen der Deployment-Descriptor-Files enthält. In Abbildung 90 wird das Initial Manifest-File für dieses Beispiel gezeigt.

Abbildung 90: Initial Manifest-File

```
Manifest-Version: 1.0  
  
Name: ServerStackDeployment.ser  
Enterprise-Bean: True  
  
Name: StackDemoDeployment.ser  
OAS-Application: True
```

6. Erstellen des ejb-jar-File

Mit dem Dienstprogramm "jar" muss ein ejb-jar-File erzeugt werden, das alle Bestandteile der Bean enthält. Dieses ejb-jar-File wird vom OAS-Manager benutzt, um die EJB-Anwendung auf dem OAS zu installieren. Dem Initial Manifest-File werden bei der Erstellung des ejb-jar-File weitere Informationen hinzugefügt. Danach wird es selbst in das ejb-jar-File gepackt.

Die Anweisung zur Erstellung eines ejb-jar-File mit dem Dienstprogramm "jar", wird in Abbildung 91 dargestellt.

Abbildung 91: Anweisung zur Erstellung eines ejb-jar-File

```
jar cmf <InitialManifestFileName> <AnwendungName.jar> *
```

Hierbei wird vorausgesetzt, dass alle Dateien im aktuellen Verzeichnis stehen. Für weitere Informationen zum Dienstprogramm "jar" sei an dieser Stelle auf dessen Hilfe verwiesen.

7. Installieren der EJB-Anwendung auf dem OAS

Zum Installieren der Anwendung auf dem OAS wird der OAS-Manager verwendet. Im OAS-Manager wird eine neue Anwendung vom Typ "Enterprise JavaBeans" erstellt und das entsprechende ejb-jar-File angegeben.

Der OAS generiert automatisch die notwendigen Dateien für die CORBA-Infrastruktur, in der die Bean angelegt wird. Generiert werden IDL-Files, Stubs und Skeletons basierend auf Home- und Remote-Interface. Diese Dateien werden kompiliert und in zwei Java-Archive mit den Namen "_client.jar" und "_server.jar" gepackt.

Diese Java-Archive werden nach der Installation in dem automatisch erstellten Verzeichnis "<OAS_Laufwerk>\<OAS_Home>\ows\apps\ejb\<AnwendungsName>\" gespeichert. Die Datei "_server.jar" wird automatisch vom CLASSPATH des OAS referenziert. Auf dem Client muss die Datei "_client.jar" gespeichert werden und vom dortigen CLASSPATH referenziert werden.

8. Erstellen des Client

Auf EJB-Objekte kann mit verschiedenen Clients zugegriffen werden. Ein Client kann eine Java-Application, ein Java-Applet oder ein Java-Servlet sein.

In Abbildung 92 ist der Quellcode eines Client (Java-Application) gezeigt, der auf die Methoden einer Bean-Klasse zugreift.

Abbildung 92: Quellcode des Client einer EJB-Anwendung

```
import javax.naming.*;
import java.util.Hashtable;
import myStack.ServerStackRemote;
import myStack.ServerStackHome;
import javax.rmi.PortableRemoteObject;
import javax.naming.*;

public class Demo
{
    public static void main(String args[])
    {
        ServerStackRemote stackRemote = null;
        String beanURL =
            "oas://pc060.nochen.opitz-partner.de:80/StackDemo/ServerStack";

        try
        {
            // Hashtable erzeugen und mit dem Namen der Klasse füllen,
            // die den Namensdienst darstellt
            Hashtable env = new Hashtable();
            env.put(Context.INITIAL_CONTEXT_FACTORY,
                "oracle.oas.naming.jndi.RemoteInitCtxFactory");
```



```
// Erzeugen des JNDI-Initial-Context für den Zugang zum
// Namensdienst
Context initialContext = new InitialContext(env);

// Auffinden des Home-Interface anhand der Bean-URL und
// Rückgabe der Referenz auf das EJBHome
ServerStackHome stackHome = (ServerStackHome)
    PortableRemoteObject.narrow(initialContext.lookup(beanURL),
        ServerStackHome.class);

// Erzeugen von Bean-Instanz und EJBObject und Rückgabe der
// Referenz auf das EJBObject
stackRemote = (ServerStackRemote) stackHome.create();

}
catch (Exception e)
{
    System.out.println("Fehler: " + e.toString());
}

try
{
    // Bestimmen der maximalen Anzahl der Stack-Elemente
    stackRemote.setStackSize(10);
    // Ausgabe der maximalen Anzahl der Stack-Elemente
    System.out.println("Stack-Größe: " +
        stackRemote.getStackSize());

    // Hinzufügen von Stack-Elementen
    stackRemote.push("Element1");
    stackRemote.push("Element2");
    stackRemote.push("Element3");

    // Herunternehmen und Ausgabe der Stack-Elemente
    System.out.println("ServerStack hatte folgende Elemente:");
    System.out.println(stackRemote.pop());
    System.out.println(stackRemote.pop());
    System.out.println(stackRemote.pop());

}
catch (java.rmi.RemoteException re)
{
    System.out.println("Remote exception " + re.toString());
    re.printStackTrace();
}

finally
{
    try
    {
        if (stackRemote != null) stackRemote.remove();
    }
    catch (java.rmi.RemoteException e) {}
    catch (RemoveException re)
    {
        System.out.println ("Remove failed");
    }
}
```

```
        System.out.println (re);  
    }  
}  
  
}
```

Wenn ein Java-Client die auf dem OAS installierte Bean benutzen möchte, so muss er die Bean zunächst finden. Dazu benutzt er einen vom OAS zur Verfügung gestellten Namensdienst, der über das Java Naming and Directory Interface (JNDI) angesprochen wird.

Um den Namensdienst ansprechen zu können, wird zuerst eine Hashtable erzeugt. In diese Hashtable wird der Konstanten `"Context.INITIAL_CONTEXT_FACTORY"` der Name der Klasse zugeordnet, die den Namensdienst darstellt. Diese Klasse wird vom Hersteller des EJB-Server (hier Oracle) mitgeliefert. Danach wird ein JNDI-Initial-Kontext mit der Hashtable als Übergabeparameter erzeugt, der für den Zugang zum Namensdienst dient.

Im nächsten Schritt wird das Home-Interface der Bean aufgefunden. Hierzu wird die Methode `"narrow"` verwendet, die als Übergabeparameter die Methode `"initialContext.lookup(beanURL)"` und die EJBHome-Klasse hat. Hierbei wird die Referenz auf das EJBHome zurückgegeben. Die Referenz auf das EJBHome wird verwendet, um eine Bean-Instanz und ein EJBObject mit der Methode `"create"` zu erzeugen. Der Container gibt die Referenz auf das EJBObject zurück. Über das EJBObject können die Methoden der Bean angesprochen werden. Im Beispiel wird die maximale Stack-Größe mit der Methode `"setStackSize"` eingestellt. Mit der Methode `"push"` werden Elemente auf den Stack gelegt und mit der Methode `"pop"` Elemente vom Stack heruntergenommen.

In der Methode `"finally"` wird die Methode `"remove"` des EJBObject aufgerufen. Dadurch wird die Bean-Instanz und das EJBObject gelöscht.

9. Client ausführen

Um den Client ausführen zu können muss zusätzlich zum Java-Archiv `"_client.jar"` das Java-Archiv `"ejbapi.jar"` und `"oasoorb.jar"` auf dem Client Rechner vorhanden sein. Der CLASSPATH muss auf diese Datei verweisen.

Die Schritte 4 bis 7 können durch Einsatz des Entwicklungswerkzeuges "JDeveloper" automatisiert werden. Zusätzlich bietet dieses Werkzeug die Möglichkeit einen Client für eine auf dem OAS installierte EJB zu erstellen.

3.7.6 Vergleich der Einsatzmöglichkeiten

In diesem Kapitel wird ein Vergleich der vorgestellten Einsatzmöglichkeiten durchgeführt. Dabei werden die Vor- und Nachteile, die Anwendungsbereiche und die Voraussetzungen betrachtet.

Mit einer PL/SQL-Anwendung bietet sich eine einfache Möglichkeit zur dynamischen Generierung von HTML-Dokumenten, die in einer Oracle-Datenbank gespeicherte Daten anzeigen. Der Datenbankzugriff wird hierbei durch einen zentral zu verwaltenden DAD realisiert. Hierzu müssen keine Datenbanktreiber verwendet werden, jedoch kann ausschließlich auf Oracle-Datenbanken zugegriffen werden.

Voraussetzung für die Entwicklung und den Einsatz einer PL/SQL-Anwendung ist, dass das PL/SQL-Toolkit in der entsprechenden Oracle-Datenbank installiert ist.

Eine Perl-Anwendung eignet sich zur Integration von bestehenden Perl-Skripten. Indem der Perl-Interpreter nach dem erstmaligen Aufruf aktiv im Speicher und eine Perl-Cartridge bereits kompiliert in einem Cache gehalten wird, bietet eine Perl-Anwendung eine gute Performance. Zudem kann über Perl-Bindings auf CORBA-Objekte zugegriffen werden. Der Datenbankzugriff aus Perl ist einfach zu realisieren, wobei jede Datenbank angesprochen werden kann, zu der ein Treiber angeboten wird.

Eine LiveHTML-Anwendung ist gegenüber einer Perl-Anwendung schneller, da ausschließlich die dynamischen Abschnitte eines HTML-Dokumentes generiert werden. Bei einer Perl-Anwendung werden zusätzlich noch die statischen Abschnitte eines HTML-Dokumentes generiert.

JServlet-Anwendungen und EJB-Anwendungen haben alle Vorteile, die Java bietet. Neben dem Datenbankzugriff über JDBC können mit Hilfe des JTS programmtechnische Transaktionen realisiert werden. Über JDBC kann jede Datenbank angesprochen werden, zu der ein Treiber angeboten wird.

C++-CORBA-Anwendungen und EJB-Anwendungen dienen nicht der Präsentation. Sie generieren keine HTML-Dokumente, sondern beinhalten ausschließlich Anwendungslogik. Für die Präsentation der Anwendung müssen hierbei explizit Client-Anwendungen entwickelt werden, welche die Anwendungslogik verwenden.

Bedingt durch die komplexe Architektur, auf der diese Anwendungstypen basieren, kann es bei der Entwicklung und verstärkt bei der Installation auf dem OAS zu Problemen kommen.

4 Beschreibung der Anwendung "OnlineTicket"

Dieses Kapitel dient als Grundlage für die Realisierung, sowie zur Einführung der Anwendung "OnlineTicket". Hierzu werden die Anforderungen, die Anwendungsarchitektur und das Datenmodell beschrieben, sowie eine Benutzeranleitung gegeben.

Anforderung

Die zu entwickelnde Anwendung soll den Vertrieb von Tickets für diverse Veranstaltungen unterstützen. Diese Veranstaltungen können an verschiedenen Veranstaltungsorten stattfinden. Jeder Veranstaltungsort hat eine bestimmte Anzahl von Plätzen, die bestimmten Preiskategorien zugeordnet sind. Jeder Preiskategorie ist, abhängig von der Veranstaltung, ein Preis zugeordnet.

Möchte ein Kunde Tickets buchen, so muss er zunächst die gewünschte Veranstaltung auswählen. Danach soll der Kunde angeben können, wieviel Tickets er für diese Veranstaltung in einer bestimmten Preiskategorie buchen möchte.

Die Anwendung soll dem Kunden die freien Plätze der Veranstaltung anzeigen und die Möglichkeit geben, diese Plätze auszuwählen. Möchte der Kunde diese ausgewählten Plätze buchen, so wird er aufgefordert seine Buchungsdaten zu erfassen.

Zwischen Auswahl der Plätze und Eingabe der Buchungsdaten sollen die ausgewählten Plätze von der Anwendung für 15 Minuten reserviert werden.

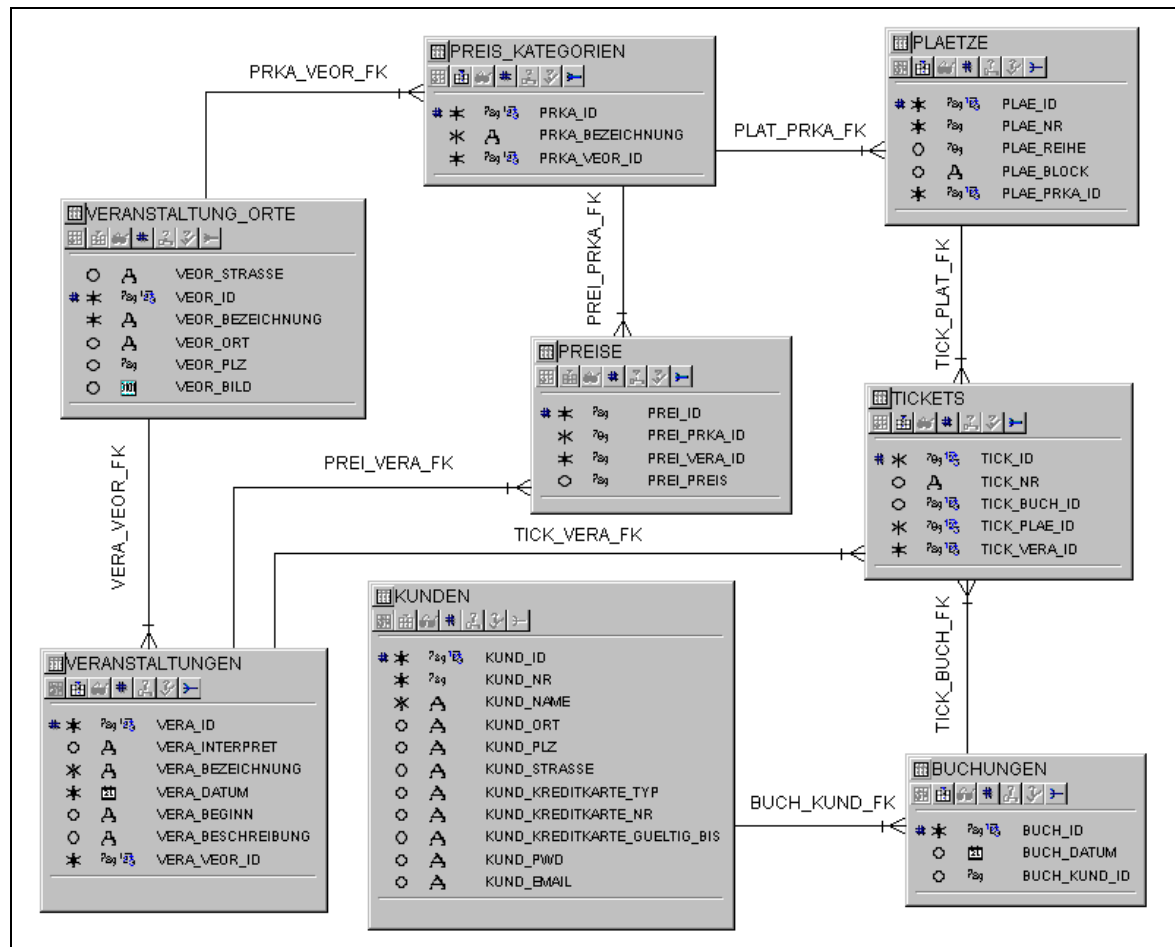
Um die reservierten Plätze zu buchen, sollen sich registrierte Kunden mit ihrer Kunden-Nr. und ihrem Passwort einloggen können. Neukunden müssen ihre gesamten Daten erfassen und sollen automatisch eine Kunden-Nr. erhalten. Werden die Buchungsdaten vom Kunden vollständig und korrekt erfasst, so soll er automatisch eine Buchungsbestätigung per E-Mail erhalten. Wird die Erfassung der Buchungsdaten abgebrochen, so sollen die reservierten Plätze von der Anwendung sofort wieder freigegeben werden.

Die Anwendung wird als Java-Anwendung realisiert. Der Benutzer kann diese ausschließlich über einen Web-Browser bedienen.

Datenmodell

In Abbildung 93 wird das Datenmodell dargestellt, auf dem die Anwendung basiert. Es zeigt die Tabellen mit ihren Attributen und ihren Beziehungen (Foreign Keys) untereinander.

Abbildung 93: Datenmodell



Im Folgenden werden die einzelnen Tabellen des Datenmodells erläutert:

- **VERANSTALTUNG_ORTE**

In dieser Tabelle werden die Orte gespeichert, in denen die Veranstaltungen stattfinden. Zu jedem Veranstaltungsort wird ein Bild gespeichert, das eine Übersicht der Preiskategorien bietet.

- VERANSTALTUNGEN

In dieser Tabelle werden alle Informationen zu einer Veranstaltung gespeichert. Neben der Bezeichnung ist das Datum, der Beginn, eine Kurzbeschreibung und der Interpret der Veranstaltung enthalten. Zudem wird ein Verweis auf den Veranstaltungsort gespeichert, an dem die Veranstaltung stattfindet.

- PREISKATEGORIEN

In dieser Tabelle werden die einzelnen Preiskategorien eines Veranstaltungsortes gespeichert.

- PREISE

In dieser Tabelle werden die einzelnen Preise der jeweiligen Preiskategorien einer Veranstaltung gespeichert.

- PLAETZE

In dieser Tabelle werden die einzelnen Plätze einer Preiskategorie gespeichert. Jeder Platz ist einer Reihe in einem Block zugeordnet.

- TICKETS

In dieser Tabelle werden die einzelnen Tickets für eine Veranstaltung gespeichert. Ein Ticket ist eine Eintrittskarte, die einen Platz für eine bestimmte Veranstaltung repräsentiert.

- BUCHUNG

In dieser Tabelle werden die einzelnen Buchungen für eine Veranstaltung gespeichert. Eine Buchung kann mehrere Tickets beinhalten. Eine Buchung ohne Verweis auf einen Kunden ist eine Reservierung. Wird einer Reservierung nicht innerhalb von 15 Minuten ein Kunde zugeordnet, so wird diese automatisch von der Anwendung gelöscht.

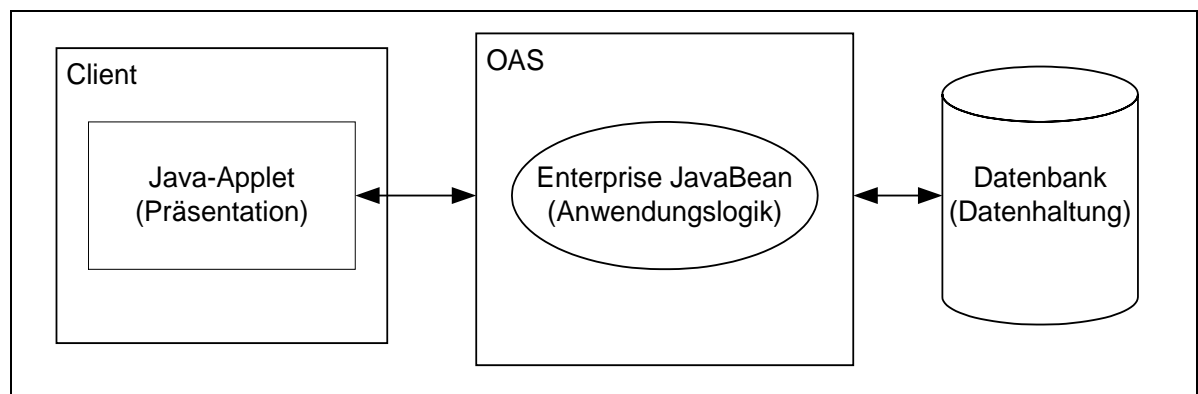
- **KUNDEN**

In dieser Tabelle werden die Kundendaten gespeichert. Neben dem Namen, der Adresse und der E-Mail-Adresse werden die Kreditkartendaten des Kunden gespeichert.

Realisierung

Die Anwendung basiert auf einer 3-Schicht-Client/Server-Architektur, wie sie in Abbildung 94 gezeigt wird.

Abbildung 94: Anwendungsarchitektur



Im Folgenden werden die drei Schichten dieser Anwendungsarchitektur mit ihren Aufgaben beschrieben:

- **Datenbank**

Die Daten der Anwendung werden in einer relationalen Oracle8-Datenbank gehalten. Die Skripte zum Erzeugen der Tabellen sind im Anhang zu finden.

- **OAS**

Der OAS bildet die Mittel-Schicht. Er stellt die Laufzeitumgebung für die Enterprise JavaBean zur Verfügung. Die EJB beinhaltet die gesamte Anwendungslogik. Diese Anwendungslogik stellt Methoden bereit, um Daten aus der Datenbank auszulesen, sie zu verarbeiten und sie in die Datenbank zu schreiben. Diese Methoden können vom Client aufgerufen werden. Der Quellcode der Enterprise JavaBean ist im Anhang zu finden.

- Client

Für den Benutzer präsentiert sich die Anwendung als Java-Applet, das ausschließlich die Präsentation der Anwendung (GUI) implementiert. Der Client baut keine direkte Verbindung zur Datenbank auf, sondern kommuniziert mit einer Enterprise JavaBean (EJB) auf dem OAS. Diese kommuniziert ihrerseits mit der Datenbank. Der Quellcode des Java-Applet ist im Anhang zu finden.

Die Realisierung der Funktionalitäten, die in den Anforderungen genannt sind, werden im Folgenden erläutert.

Versand der Buchungsbestätigung als E-Mail:

Nachdem der Kunde die Buchungsdaten vollständig erfasst hat, wird ihm eine Buchungsbestätigung in Form einer E-Mail zugesendet. Hierfür ruft der Client die Methode `"send_mail(Autor, Empfänger, Betreff, Text)"` der EJB auf. Die Methode erstellt die zu versendende Buchungsbestätigung und speichert sie in einer Textdatei. Diese Textdatei wird mit dem E-Mail-Programm "qmail" auf Kommandoebene zum Empfänger versendet.

Löschen der Reservierungen:

Zwischen Auswahl der Plätze und Eingabe der Buchungsdaten werden die ausgewählten Plätze von der Anwendung für 15 Minuten für den Kunden reserviert. Hierfür legt die Anwendung eine Reservierung an, nachdem der Kunde die Plätze ausgewählt hat. Diese Reservierung ist ein Datensatz in der Tabelle Buchungen, ohne Verweis auf einen Kunden. In diesem Buchungsdatensatz wird bei dessen Erzeugung ein Zeitstempel abgespeichert. Die PL/SQL-Prozedur `"delete_Reservierungen"` überprüft diesen Zeitstempel und löscht automatisch alle Buchungsdatensätze, die älter als 15 Minuten sind. Diese Prozedur wird von einem DBMS-Job jede Minute ausgeführt.

Der Quellcode der PL/SQL-Prozedur `"delete_Reservierungen"` ist im Anhang zu finden.

Bilder aus der Datenbank abfragen:

Nachdem der Kunde eine Veranstaltung ausgewählt hat, gelangt er in die Maske zur Platzauswahl. Hier wird dem Kunden ein Bild mit der Platzverteilung des Veranstaltungsortes angezeigt. Dieses Bild wird in der Datenbank mit dem Datentyp BLOB gespeichert. Um das Bild in der Maske anzuzeigen, ruft der Client die Methode "getVeranstaltungsortBild" der EJB auf. Die EJB selektiert das Bild aus der Datenbank und gibt es als Byte-Array an den Client zurück. Mit dem Byte-Array erzeugt der Client ein Image, welches in der Maske angezeigt wird.

Benutzeranleitung

Um ein Ticket für eine Veranstaltung zu buchen, muss der Benutzer folgende Schritte durchführen:

1. Veranstaltung auswählen
2. Platzwahl
3. Buchungsdaten erfassen

Diese einzelnen Schritte werden anhand eines Beispiels beschrieben. In diesem Beispiel sollen für die Veranstaltung "Bläck Fööss Silvesterparty 2000" am 31.12.2000 vier Tickets gebucht werden. Diese Plätze sollen sich nebeneinander in einer Reihe befinden. Der Betrag soll von der VisaCard abgebucht werden.

1. Veranstaltung auswählen

In Abbildung 95 wird die Maske zur Auswahl der Veranstaltung dargestellt. Sie ist in drei Bereiche aufgeteilt.

Im Bereich 1 können Kriterien für die Suche nach Veranstaltungen eingegeben werden. Es kann nach Bezeichnung, Ort, Interpret und / oder Datum gesucht werden. Zusätzlich kann das Zeichen "%" als Wildcard benutzt werden.

Abbildung 95: Maske "Veranstaltung auswählen"

OnlineTicket - Veranstaltung auswählen

Bezeichnung

Ort **1**

Interpret

Datum von bis

Suchen

3

Bläck Fööss Silvesterparty 2000
 Datum: 31.12.2000
 Ort: Kölnarena
 Beginn: 21:00 Uhr

Sie haben gezeigt, wie eine richtige Silvesterparty aussieht. Sie haben in Deutschlands größter und modernster Veranstaltungshalle vor ausverkauftem Haus das neue Jahrtausend eingeläutet und ihren 30. Geburtstag gefeiert. Sie kommen wieder: Die Bläck Fööss. Wie im vergangenen Jahr werden die sieben Fööss Kafi Biermann, Ralph Gusovius, Bömmel Lückerrath, Hartmut Priess, Willy Schnitzler, Peter Schütten und Erry Stoklosa auch dann wieder bis weit nach Mitternacht auf der Bühne stehen. Silvester 2000 wird ein rauschendes Familienfest mit den Bläck Fööss in der Kölnarena.

ID	Bezeichnung	Datum	Ort
14	Rundfunkchor Berlin	10.06.2000	St. Maria im Kapitol
24	Innercity	10.06.2000	Kölnarena
15	Münchner Philharmoniker mit Ja...	11.06.2000	Kölner Philharmonie
16	O Day Dances	17.06.2000	Schauspielhaus
12	Savion Glover "Foot Notes"	22.06.2000	Kölner Philharmonie
19	Elton John	27.06.2000	Kölnarena
13	Savion Glover "Foot Notes"	29.06.2000	Kölner Philharmonie
26	Saturday Night Fever	20.08.2000	Musical Dome Köln
34	Stars 2000	17.09.2000	Kölnarena
11	Chris Rea 2000	24.09.2000	Kölnarena
27	Höhner-Classic	26.09.2000	Kölner Philharmonie
28	Reinhard Mey - Einhandsegler	25.10.2000	Kölner Philharmonie
22	Udo Jürgens 2000	12.11.2000	Kölnarena
18	Britney Spears Live in Concert 20...	13.11.2000	Kölnarena
37	Mövenpick Art on Ice	30.11.2000	Kölnarena
23	PUR	13.12.2000	Kölnarena
35	PUR	14.12.2000	Kölnarena
29	Nokia Night of the Proms 2000	16.12.2000	Kölnarena
20	Maffay Tour 2000	17.12.2000	Kölnarena
25	Helmut Lotti	21.12.2000	Kölnarena
17	Bläck Fööss Silvesterparty 2000	31.12.2000	Kölnarena
30	André Rieu	06.01.2001	Kölnarena
21	DJ Bobbo	24.01.2001	Kölnarena
36	Musikantenstadl	01.02.2001	Kölnarena
33	OLDIE TOTAL	16.03.2001	Kölnarena
31	Die Flippers	17.03.2001	Kölnarena

Bestellen

Im Bereich 2 werden die Veranstaltungen, die den Suchkriterien entsprechen, in einer Tabelle nach Datum geordnet angezeigt. Wird eine Veranstaltung markiert, so wird eine Kurzbeschreibung dazu im Bereich 3 dargestellt und die Schaltfläche "Bestellen" aktiviert. Durch Drücken dieser Schaltfläche gelangt man zur Maske "Platzwahl".

2. Platzwahl

In Abbildung 96 wird die Maske für die Platzwahl gezeigt. Sie ist in vier Bereiche aufgeteilt.

Im Bereich 1 kann die Anzahl der Tickets und die Preiskategorie dieser Tickets ausgewählt werden. Um die Auswahl abzuschließen und die freien Plätze zu ermitteln, muss die Schaltfläche "Plätze suchen" gedrückt werden. Die freien Plätze werden im Bereich 2 in einer Tabelle nach Preiskategorie geordnet angezeigt. Neben der Preiskategorie wird der

Preis, der Block und die Reihe der Plätze angezeigt. In der Spalte "frei" wird die Anzahl der freien Plätze der entsprechenden Reihe angezeigt.

Abbildung 96: Maske "Platzwahl"

OnlineTicket - Platzwahl

Tickets: 4 Preiskategorie: [1, Unterang, 120 DM]

Plätze suchen

Preiskategorie	Preis	Block	Reihe	frei
Unterang	120	204	1	9
Unterang	120	204	2	8
Unterang	120	204	6	4
Unterang	120	204	7	4
Unterang	120	204	10	5
Unterang	120	205	3	4
Unterang	120	205	5	5
Unterang	120	205	6	5
Unterang	120	205	9	5
Unterang	120	205	10	10
Unterang	120	206	1	10
Unterang	120	206	2	7
Unterang	120	206	5	10
Unterang	120	206	7	4
Unterang	120	206	8	10
Unterang	120	206	9	10
Unterang	120	207	1	5
Unterang	120	207	2	10
Unterang	120	207	3	10
Unterang	120	207	4	10
Unterang	120	207	5	10
Unterang	120	207	6	10

OK Abbrechen

Bläck Fööss Silvesterparty 2000

Datum: 31.12.2000 Ort: Kölnarena Beginn: 21:00 Uhr

Das Parkett ist variabel bestuhbar von 1 - 12 Blöcken.
Bitte erkundigen Sie sich bei Ihrer persönlichen VVK-Stelle.

Bühne

Im Bereich 3 wird der Name, das Datum, der Ort und der Beginn der Veranstaltung angezeigt.

Im Bereich 4 wird der Veranstaltungsort mit seinen Blöcken und Preiskategorien graphisch dargestellt.

Wird eine Zeile in der Tabelle markiert, so wird die Schaltfläche "OK" aktiviert. Durch Drücken dieser Schaltfläche gelangt man zur Maske "Buchungsdaten erfassen".

3. Buchungsdaten erfassen

In Abbildung 97 wird die Maske zum Erfassen der Buchungsdaten dargestellt. Sie ist in zwei Bereiche aufgeteilt.

Abbildung 97: Maske "Buchungsdaten erfassen"

OnlineTicket - Buchungsdaten erfassen

Kunden-Nr.

Passwort

Buchungsinformationen:

Veranstaltung:
Bläck Fööss Silvesterparty 2000
Datum: 31.12.2000 Ort: Kölnarena Beginn: 21:00 Uhr

Plätze:
01. Block: 205 Reihe: 006 Platz: 006 Preis: 120,00 DM (Unterang)
02. Block: 205 Reihe: 006 Platz: 007 Preis: 120,00 DM (Unterang)
03. Block: 205 Reihe: 006 Platz: 008 Preis: 120,00 DM (Unterang)
04. Block: 205 Reihe: 006 Platz: 009 Preis: 120,00 DM (Unterang)

zzgl. Bearbeitungsgebühren 4,97 DM

Gesamtpreis 484,97 DM

Im Bereich 1 werden Informationen zur ausgewählten Veranstaltung und die ausgewählten Plätze angezeigt. Zudem wird in diesem Bereich die Bearbeitungsgebühr, sowie der zu zahlende Gesamtpreis ausgegeben.

Im Bereich 2 gibt es zwei Möglichkeiten. Zum einen kann sich ein registrierter Kunde mit seiner Kunden-Nr. und seinem Passwort einloggen und die Buchung durch Drücken der Schaltfläche "Buchten" abschließen.

Zum anderen kann sich ein Neukunde durch Drücken der Schaltfläche "Neuer Kunde" registrieren. Dazu wird die Maske um weitere Felder ergänzt, in die der Kunde persönliche Daten eintragen muss. In Abbildung 98 werden diese zusätzlichen Felder gezeigt.

Nach dem Drücken der Schaltfläche "Buchten" wird dem Neukunden automatisch eine Kunden-Nr. zugeordnet. Der Kunde wird in der Datenbank gespeichert und die Buchung abgeschlossen.

Abbildung 98: Maske "Buchungsdaten erfassen" (Neukunde)

OnlineTicket - Buchungsdaten erfassen

Kunden-Nr.

wird automatisch vergeben

Passwort

Nach-, Vorname

Hüskes, Oliver

Straße

Dellbrücker Str. 104

PLZ

51469

Ort

Bergisch Gladbach

E-Mail

hueskes@opitz-partner.de

Zahlungsart

VisaCard

Karten-Nr.

98893456122345

Karte gueltig bis

05 / 03

Reset

Buchen

Abbrechen

Buchungsinformationen:

Veranstaltung:

Bläck Fööss Silvesterparty 2000

Datum:

31.12.2000

Ort:

Kölnarena

Beginn:

21:00 Uhr

Plätze:

01. Block: 205 Reihe: 006 Platz: 006 Preis: 120,00 DM (Unterang)

02. Block: 205 Reihe: 006 Platz: 007 Preis: 120,00 DM (Unterang)

03. Block: 205 Reihe: 006 Platz: 008 Preis: 120,00 DM (Unterang)

04. Block: 205 Reihe: 006 Platz: 009 Preis: 120,00 DM (Unterang)

zzgl. Bearbeitungsgebühren

4,97 DM

Gesamtpreis

484,97 DM

Die Eingabe der Daten kann jederzeit über das Drücken der Schaltfläche "Reset" verworfen werden. Die Maske erhält daraufhin das Aussehen der Abbildung 97.

Durch Betätigen der Schaltfläche "Abbrechen" gelangt man wieder zur Maske "Veranstaltung auswählen".

Nachdem eine Buchung erfolgreich abgeschlossen wurde, wird dies dem Benutzer durch ein Bestätigungsfenster signalisiert. Der Benutzer erhält eine Buchungsbestätigung per E-Mail, sofern er eine E-Mail-Adresse angegeben hat und die Tickets werden ihm per Post zugeschickt.

In Abbildung 99 wird die Buchungsbestätigung für die in den vorangegangenen Abbildungen gezeigte Buchung dargestellt.

Abbildung 99: E-Mail "Buchungsbestätigung"

```
Subject:    Buchungsbestätigung OnlineTicket
Date:      Mon, 26 Jun 2000 18:30:20 +0200
From:      service@OnlineTicket.de
To:        hueskes@opitz-partner.de

Sehr geehrter Kunde,

wir bedanken uns für Ihre Buchung und übersenden Ihnen hiermit
folgende Buchungsbestätigung:

Veranstaltung:
Bläck Fööss Silvesterparty 2000
Datum: 31.12.2000      Ort: Kölnarena      Beginn: 21:00 Uhr
Buchungsnummer: 1177
Kunden-Nr: 1396

Plätze:
01. Block: 205 Reihe: 006 Platz: 006 Preis: 120,00 DM (Unterang)
02. Block: 205 Reihe: 006 Platz: 007 Preis: 120,00 DM (Unterang)
03. Block: 205 Reihe: 006 Platz: 008 Preis: 120,00 DM (Unterang)
04. Block: 205 Reihe: 006 Platz: 009 Preis: 120,00 DM (Unterang)

zzgl. Bearbeitungsgebühren                      4,97 DM
-----
Gesamtpreis                                     484,97 DM

Der Gesamtpreis von 484,97 DM wird von Ihrer Kreditkarte abgebucht.
Wir wünschen Ihnen eine erlebnisreiche Veranstaltung.

Ihr OnlineTicket-Team

OnlineTicket GmbH
Kölner Str. 78a
50678 Köln
Tel.: 0221/600151
```

Entwicklungstools

Das Datenmodell wurde mit dem CASE-Tool Oracle Designer 6.0 entwickelt. Das Java-Applet und die Enterprise JavaBean wurden mit der integrierten Entwicklungsumgebung Oracle JDeveloper 3.1 entwickelt.

5 Schlußbetrachtung

Der Oracle Application-Server ist eine zentrale Plattform für den Einsatz von Anwendungen im WWW oder im Intranet. Diese Anwendungen können in unterschiedlichen Programmiersprachen entwickelt werden. Die Benutzer können diese über einen Web-Browser aufrufen. Die Ausführung der Anwendungen ist somit betriebssystemunabhängig und stellt an die client-seitige Hardware minimale Anforderungen.

Beim Einsatz der Anwendungen über einen Application-Server werden Middleware-Funktionen wie Skalierbarkeit, Zuverlässigkeit und Sicherheit bereitgestellt, ohne dass der Entwickler diese Aspekte bei der Anwendungsentwicklung explizit berücksichtigen muss. Die Middleware-Funktionen werden durch den OAS als Ganzes installiert und verwaltet. Das ist wesentlich kosteneffizienter und weniger kompliziert, als eine Sammlung unterschiedlichster Middleware-Produkte zu integrieren und zu verwalten.

Der OAS-Manager ist als web-basiertes Werkzeug im Lieferumfang des OAS enthalten. Damit kann die Administration, Konfiguration, Wartung und Überwachung des OAS sehr komfortabel durchgeführt werden. Hiermit hat der Administrator die Möglichkeit den OAS entfernt über das WWW oder Intranet zu administrieren.

Der OAS stellt mit dem Database Access Descriptor (DAD) eine einfache Möglichkeit zur Verfügung, um aus dem Internet auf eine Oracle-Datenbank zugreifen zu können. Ein DAD kann aus einer PL/SQL-Anwendung oder aus einer Java-Anwendung über JDBC angesprochen werden. Die Parameter zur DB-Verbindung sind im DAD zentral hinterlegt und müssen nicht in jeder Anwendung erneut spezifiziert werden. Der Datenbankzugriff ist dadurch gekapselt, so daß bei Änderungen der Parameter nicht die Anwendungslogik überarbeitet werden muss.

Weiterhin können mit dem Dienstprogramm "pl2java" Stored-Procedures in Java-Anwendungen verwendet werden. Das heißt, die Anwendungslogik einer Anwendung kann sowohl Java-Code als auch PL/SQL-Code enthalten. Somit können PL/SQL-Packages ohne jegliche Änderung wiederverwendet werden.

Von Vorteil ist, dass die Enterprise Edition des OAS verschiedene Interaktionsmodelle wie Sitzungen und Transaktionen, sowie den Datenbankzugriff über ODBC unterstützt. Allerdings erscheint die Enterprise Edition deutlich überteuert, da Oracle dafür den doppelten Preis der Standard Edition verlangt.

Der OAS unterstützt verschiedene Sicherheitsaspekte. Es können sowohl cartridge-basierte als auch komponenten-basierte Anwendungen abgesichert werden. Unterstützt werden Benutzer- und Host-Authentifizierung. In den Benutzer-Authentifizierungsschemata können die Benutzer einer Oracle-Datenbank herangezogen werden. Zusätzlich unterstützt der OAS die Datenverschlüsselung mit SSL, einem der bedeutensten Verschlüsselungsverfahren.

Eine weitere, immer bedeutender werdende Funktion des OAS ist die Unterstützung der Komponentenmodelle CORBA und Enterprise JavaBeans. Die Unterstützung dieser Komponentenmodelle ist zukunftsorientiert, da sich die komponentenbasierte Anwendungsentwicklung in zunehmendem Maße durchsetzt und verbreitet. Wünschenswert in der zukünftigen Version des OAS wäre allerdings die Unterstützung der EJB-Spezifikation 1.1 und die Möglichkeit zusätzlich zu Session-Beans auch Entity-Beans einsetzen zu können.

Es wäre praktisch, wenn die Unterstützung der einzelnen Anwendungstypen modulweise installiert und migriert werden kann. Möchte man z.B. von der EJB-Version 1.0 auf die EJB-Version 1.1 migrieren, so kann man nur eine komplett neue Version des OAS beschaffen und installieren.

Bei der Entwicklung der Anwendung "OnlineTicket" erwies sich der JDeveloper von Oracle als geeignetes Werkzeug, um Java-Anwendungen zu erstellen. Vor allem ist das einfache Entwickeln und Testen von EJBs positiv herauszustellen. Da der OAS mit dem JDeveloper kooperiert, werden dem Entwickler beim Installieren der EJBs auf dem OAS viele Schritte abgenommen.

Positiv zu bewerten ist das Failure Recovery des OAS. Dadurch wird verhindert, dass eine einzelne Fehlerquelle den gesamten OAS zum Absturz bringen kann, wodurch kostenverursachende Ausfallzeiten minimiert werden.

Bei der Arbeit mit dem OAS zeigte sich, dass er sehr zuverlässig arbeitet. Wurden allerdings nach der Installation des OAS weitere Oracle-Produkte auf dem Server installiert, so kam es zu Problemen, die sich hauptsächlich auf die Einrichtung von EJBs auswirkten.

In der Vorbereitung und während des Verfassens dieser Diplomarbeit gestaltete sich die Literatursuche schwierig, da zum Thema "Application-Server" kaum Literatur existiert. Hierbei ist zu bemerken, dass sich als Informationsquelle und bei Problemen die Internet-Diskussionsforen von Oracle als sehr hilfreich erwiesen haben.

Zum Abschluss möchten wir uns bei der Firma Opitz & Partner GmbH für ihre Unterstützung während unserer Diplomarbeit bedanken.

ANHANG

Glossar

CA - Certificate authority

Eine CA ist eine Einrichtung, die Zertifikate an Einzelpersonen oder Unternehmen nur dann ausgibt, wenn es die Einzelperson bzw. das Unternehmen überprüft hat.

Cookie

Ein Cookie ist eine zwischen Client und Server ausgetauschte Information, die zum Aufrechterhalten der Information über den Zustand dient. Die Cookies werden vom Server client-seitig gespeichert und bleiben bis zu einem vom Server angegebenen Verfallsdatum gültig.

CORBA - Common Object Request Broker Architecture

CORBA ist eine von der Object Management Group (→ OMG) erarbeitete Spezifikation, welche die Zusammenarbeit von Softwaresystemen über die Grenzen von Programmiersprachen, Betriebssystemen und Netzwerken hinweg regelt.

DAD - Data Access Descriptor

Ein Data Access Descriptor ist eine Ansammlung von Informationen, die der OAS benötigt, um eine Verbindung zwischen einer PL/SQL-Cartridge und einer Oracle-Datenbank herzustellen.

EJB – Enterprise JavaBeans

Enterprise JavaBeans ist ein Komponentenmodell zum Erstellen von Java-Anwendungen in einer verteilten Umgebung.

Gartner Group

Die Gartner Group ist weltweit eines der führenden Beratungsunternehmen im IT-Umfeld. Sie erstellt hauptsächlich Berichte und Analysen über Produkte und Technologien, die verschiedene IT-Bereiche abdecken.

GIOP - General Inter-ORB Protocol

Das General Inter-ORB Protocol spezifiziert eine Menge von Nachrichtenformaten und Datendarstellungen für die Kommunikation zwischen → ORBs. GIOP definiert außerdem ein Format für Interoperable Object References (→ IOR).

ICX – Inter Cartridge Exchange

Ein Dienst, der es einer Cartridge ermöglicht, eine andere Cartridge aufzurufen und dessen Funktionalität zu nutzen.

IDL – Interface Definition Language

Eine von der → OMG definierte Sprache zur programmiersprachenunabhängigen Beschreibung von Objektschnittstellen.

IIOP – Internet Inter-ORB Protocol

Das in CORBA am häufigsten verwendete Kommunikations-Protokoll auf TCP/IP-Basis. Es legt fest, wie GIOP-Nachrichten über ein TCP/IP-Netzwerk ausgetauscht werden.

IOR – Interoperable Object Reference

ORB-übergreifende Objektreferenz eines CORBA-Objektes, die weltweit eindeutig ist. Sie beinhaltet die IP-Adresse des Rechners, auf dem der ORB installiert ist, den Port des ORB und eine ORB-interne Identifikation.

JDBC - Java Database Connectivity

JDBC ist ein in Java implementiertes API, das Klassen, Interfaces und Exceptions bereitstellt, um SQL-Anweisungen an ein Datenbankmanagementsystem (DBMS) zu senden und auszuwerten.

JNDI - Java Naming and Directory Interface

JNDI ist ein in Java implementiertes API, das Klassen, Interfaces und Exceptions bereitstellt, um aus Java-Anwendungen auf einen Namens- und Verzeichnisdienst zugreifen zu können.

JTA - Java Transaction API

→ JTS

JTS - Java Transaction Service

JTS ist ein in Java implementiertes API, das Klassen, Interfaces und Exceptions bereitstellt, um Transaktionen realisieren zu können.

JVM – Java Virtual Machine

Java wird nicht in Maschinencode übersetzt, sondern in einen plattformunabhängigen, Java-Bytecode umgewandelt, der auf dem jeweiligen Rechner von einem Java-Interpreter, der Java Virtual Machine, ausgeführt wird.

Komponentenmodell

Legt die Grundlagen für die Zusammenarbeit von Software-Bausteinen fest.

Object Adapter

Der Object Adapter bietet den grundsätzlichen Zugriff für die Objektimplementierung auf Dienste, die der ORB anbietet..

ODBC – Open Database Connectivity

Standard für den Datenbankzugriff, der von Microsoft entwickelt worden ist. Für die verschiedenen Datenquellen werden von den Herstellern entsprechende ODBC-Treiber angeboten.

OMG - Object Management Group

Die OMG setzt sich aus über 800 Herstellern und Anwendern zusammen. Die OMG fördert die Theorie und Praxis der objektorientierten Technologie in der Softwareentwicklung. Ziel der OMG ist die Erstellung von Industrienormen und Spezifikationen zur Unterstützung einer allgemeinen Plattform zur Anwendungsentwicklung. <http://www.omg.org>

Oracle Wallet Manager

Der Oracle Wallet Manager ist eine Java-Anwendung für die Verwaltung von Zertifikaten. Mit seiner Hilfe können Schlüsselpaare (Public-Key / Private-Key) und Zertifikatsanforderungen generiert werden, und die erhaltenen Zertifikate auf dem OAS installiert werden.

ORB – Object Request Broker

Mit der Unterstützung eines ORB in einem Netzwerk hat ein Client die Möglichkeit eine Anfrage an ein entferntes Objekt zu stellen, ohne Wissen darüber zu haben, wo sich dieses entfernte Objekt im Netzwerk befindet und in welcher Sprache es implementiert ist.

pl2java

Dienstprogramm zur Generierung von Java-Wrapper-Klassen. Mit den generierten Wrapper-Klassen können PL/SQL-Funktionen oder PL/SQL-Prozeduren aus Java-Anwendungen heraus aufgerufen werden.

RM-Proxy

Erhält die Referenzen auf EJB- und C++-Objekte und gibt diese zurück an die Clients.

Skeleton

Skeletons stellen das server-seitige Gegenstück zu → Stubs dar. Sie leiten die Aufrufe der Clients an die Objektimplementierung weiter und geben auf umgekehrten Weg das Ergebnis des Aufrufes an den Client zurück.

Socket

Die Kombination einer IP-Adresse mit einer Port-Nummer wird Socket genannt.

SSL - Secure Sockets Layer

Ein Standard für die sichere Übertragung von Dokumenten über das Internet mit HTTPS.

Stub

Ein Stub hat die Aufgabe einen Methodenaufruf an ein entferntes Objekt weiterzuleiten und dessen Antwort an den Client zurückzugeben.

Thin-Client

Ein Rechner auf dem nur ein Web-Browser und optional ein JVM installiert ist. Die Anwendungen werden nicht installiert, sondern mittels eines Web-Browsers runtergeladen und bei Bedarf von einer Java Virtual Machine ausgeführt.

WRB - Web Request Broker

Der WRB ist ein OAS-Runtime-Prozess, der die Ressourcen des OAS verwaltet. Bei der Multi-Node-Konfiguration teilen sich der Primary-Node und die Remote-Nodes denselben WRB.

Quellcode des Java-Applet

Quellcode der Klasse "AppletTicketShop"

```
//=====
// Klasse:      AppletTicketShop
// Projekt:      OnlineTicket
// Version:      1.0 vom 23.06.2000
// Autor:        Oliver Hüskes & Michael Zeidler
// Beschreibung:
//=====
package client;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import javax.swing.*;
import java.sql.*;
import java.util.Hashtable;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import javax.naming.CommunicationException;

public class AppletTicketShop extends JApplet {
    boolean isStandalone = false;

    // HomeInterface der Bean
    oneEJBTicket.TicketHome homeInterface = null;

    // RemoteInterface der bean
    oneEJBTicket.TicketRemote remoteTicketEJB;

    // Panel VeranstaltungSuchen einrichten
    PanelVeranstaltungAuswaehlen paVeranstaltungAuswaehlen =
        new PanelVeranstaltungAuswaehlen(this);

    // Platzhalter für PanelPlatzwahl
    PanelPlatzwahl paPlatzwahl ;

    // Platzhalter für PanelBuchungsdatenErfassen
    PanelBuchungsdatenErfassen paBuchungsdatenErfassen ;

    // InitalContext
    Context ic;

    //-----Konstruktor-----
    public AppletTicketShop() {
    }

    //-----getBeanInstance-----
    // fordert beim OAS eine neue Bean-Instance über JNDI an
    public void getBeanInstance() {

        // URL zum Auffinden der Bean
        String ejbUrl = "oas:///TicketApp/TicketRemote";
        // String ejbUrl = "oas://pc090.nochen.opitz-
partner.de:80/TicketApp/TicketRemote";

        // Hashtable erzeugen
```

```
Hashtable environment = new Hashtable();

// Umgebungsvariablen für den JNDI-Context setzen
environment.put("oracle.oas.naming.jndi.appletinstance",this);
environment.put(Context.INITIAL_CONTEXT_FACTORY,
    "oracle.oas.naming.jndi.RemoteInitCtxFactory");
environment.put(Context.SECURITY_PROTOCOL,"AM_ORACLE_DEFAULT");
environment.put(Context.SECURITY_PRINCIPAL, "EJBUser");
environment.put(Context.SECURITY_CREDENTIALS, "EJBUser");

try {
    System.out.println("Erzeugen des initialContext");
    ic = new InitialContext(environment);

    System.out.println("Suchen nach der EJB: TicketApp/TicketRemote");
    homeInterface = (oneEJBTicket.TicketHome)
        javax.rmi.PortableRemoteObject.narrow(ic.lookup(ejbUrl),
            oneEJBTicket.TicketHome.class);
}
catch (CommunicationException e) {
    System.out.println("Communication exception!  Unable to connect: " +
        ejbUrl);
    e.printStackTrace();
    System.exit(1);
}
catch (NamingException e) {
    System.out.println("Naming exception occurred!");
    e.printStackTrace();
    System.exit(1);
}

// BeanInstance erzeugen
// Als Rückgabewert erhält man das RemoteInterface der Bean
try {
    System.out.println("Neue EJBInstance erzeugen");
    remoteTicketEJB = homeInterface.create();
}
catch (Exception e) {
    System.out.println(e.getMessage());
    e.printStackTrace();
}
}

//----- Panel panelPlatzwahl erzeugen und anzeigen -----
public void panelPlatzwahlAktivieren(String pVeraID) {

    // Panel VeranstaltungSuchen verstecken
    paVeranstaltungAuswaehlen.setVisible(false);

    // Panel panelPlatzwahl erzeugen und anzeigen
    paPlatzwahl = new PanelPlatzwahl(this,pVeraID);

    this.getContentPane().add(paPlatzwahl, null);
}

//----- Panel PanelPlatzwahl löschen -----
```



```
public void panelPlatzwahlLoeschen() {
    // Panel VeranstaltungSuchen anzeigen
    paVeranstaltungAuswaehlen.setVisible(true);

    // Panel PanelBuchungsdatenErfassen löschen
    this.getContentPane().remove(paPlatzwahl);
}

//----- Panel PanelBuchungsdatenErfassen erzeugen und anzeigen -----
public void panelBuchungsdatenErfassenAktivieren(String pVeraID,
                                                String pBuchID) {

    // Panel VeranstaltungSuchen und PanelPlatzwahl verstecken
    paVeranstaltungAuswaehlen.setVisible(false);
    paPlatzwahl.setVisible(false);

    // Panel PanelBuchungsdatenErfassen erzeugen und anzeigen
    paBuchungsdatenErfassen =
        new PanelBuchungsdatenErfassen(this, pVeraID, pBuchID);
    this.getContentPane().add(paBuchungsdatenErfassen, null);
}

//----- Panel PanelBuchungsdatenErfassen löschen -----
public void panelBuchungsdatenErfassenLoeschen() {
    // Panel VeranstaltungSuchen anzeigen
    paVeranstaltungAuswaehlen.setVisible(true);

    // Panel PanelBuchungsdatenErfassen löschen
    this.getContentPane().remove(paBuchungsdatenErfassen);
}

//----- init() -----
public void init() {
    // Die Methode init wird einmal beim Laden des Applets aufgerufen
    // hier wird das Applet initialisiert
    this.getContentPane().setLayout(null);

    // Größe des Applets bestimmen
    this.setSize(800,600);

    // EJB Instanz erzeugen
    getBeanInstance();

    // Panel VeranstaltungSuchen dem Applet hinzufügen
    this.getContentPane().add(paVeranstaltungAuswaehlen, null);
}

//----- destroy() -----
// Die Methode destroy wird aufgerufen, wenn der Browser geschlossen wird
// hier werden vom Applet benutzte Ressourcen wieder freigegeben
public void destroy() {
    try {
        // Löschen der Bean-Instanz
        if (remoteTicketEJB!=null) remoteTicketEJB.remove();
        // Schließen des InitialContext
        ic.close();
    }
    catch (Exception e) {
```

```

        e.printStackTrace();
    }
}

//-----Ende der Klasse AppletTicketShop-----

```

Quellcode der Klasse "PanelVeranstaltungAuswaehlen"

```

//=====
// Klasse:      PanelVeranstaltungAuswaehlen
// Projekt:      OnlineTicket
// Version:      1.0 vom 23.06.2000
// Autor:        Oliver Hüskes & Michael Zeidler
// Beschreibung: Maske zur Auswahl einer Veranstaltung
//=====

package client;

import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.text.*;

public class PanelVeranstaltungAuswaehlen extends JPanel {

    // Referenz auf das Startapplet AppletTicketShop
    AppletTicketShop vMasterApplet ;

    // Überschrift des Panels
    JLabel lbUeberschrift = new JLabel();

    // Konstanten die Zeilen und Spaltengröße definieren
    int vZeile = 20;
    int vSpalte = 10;

    // Komponenten des Panels
    // Bezeichnung
    JTextField tfVeraBezeichnung = new JTextField();
    JLabel lbVeraBezeichnung = new JLabel();

    // Ort
    JTextField tfVeraOrt = new JTextField();
    JLabel lbVeraOrt = new JLabel();

    // Datum
    JTextField tfVeraDatumVon = new JTextField();
    JTextField tfVeraDatumBis = new JTextField();
    JLabel lbVeraDatumVon = new JLabel();
    JLabel lbVeraDatumBis = new JLabel();

    // Interpret
    JTextField tfVeraInterpret = new JTextField();
    JLabel lbVeranstaltungInterpret = new JLabel();
}

```

```
// JTable zur Anzeige der Veranstaltungen
JTable      taVeranstaltungen ;
genTableModel tmVeranstaltungen; // TableModel der JTable
JScrollPane spVeranstaltungen ;// Scrollpane der JTable

// Informationen zur selektierten Veranstaltung
JTextPane tpVeranstaltungInfo = new JTextPane();
JScrollPane spVeranstaltungInfo = new JScrollPane() ;

//Buttons
JButton btSuchen          = new JButton();
JButton btBestellen       = new JButton();

//-----Konstruktor-----
public PanelVeranstaltungAuswaehlen(AppletTicketShop pMasterApplet ) {

    // Referenz des MasterApplets speichern
    vMasterApplet= pMasterApplet;
    // Panel initialisieren
    jbInit();
}

//-----Initialisierung des Panel-----
private void jbInit() {
    // JTable initialisieren
    init_taVeranstaltungen();

    // Grösse des Panels bestimmen
    this.setSize(800,600);
    this.setLayout(null);

    //Bezeichnung
    tfVeraBezeichnung.setBounds(new Rectangle(vSpalte * 10, vZeile*4,
                                                vSpalte * 22, vZeile));
    this.add(tfVeraBezeichnung, null);
    lbVeraBezeichnung.setText("Bezeichnung");
    lbVeraBezeichnung.setBounds(new Rectangle(20, vZeile*4,
                                                vSpalte * 10, vZeile));
    lbVeraBezeichnung.setForeground(Color.black);
    this.add(lbVeraBezeichnung, null);

    // Ort
    tfVeraOrt.setBounds(
        new Rectangle(vSpalte*10,vZeile*6,vSpalte*22,vZeile));
    lbVeraOrt.setText("Ort");
    lbVeraOrt.setBounds(new Rectangle(20, vZeile*6, vSpalte * 10, vZeile));
    lbVeraOrt.setForeground(Color.black);

    this.add(tfVeraOrt, null);
    this.add(lbVeraOrt, null);

    // Interpret
    tfVeraInterpret.setBounds(new Rectangle(vSpalte * 10, vZeile*8,
                                                vSpalte * 22, vZeile));
    lbVeranstaltungInterpret.setText("Interpret");
    lbVeranstaltungInterpret.setBounds(new Rectangle(20, vZeile*8,
                                                vSpalte * 10, vZeile));
    lbVeranstaltungInterpret.setForeground(Color.black);
    this.add(tfVeraInterpret, null);
    this.add(lbVeranstaltungInterpret, null);
}
```

```
// Datum
lbVeraDatumVon.setText("Datum    von");
lbVeraDatumVon.setForeground(Color.black);
lbVeraDatumVon.setBounds(
    new Rectangle(20,vZeile*10,vSpalte * 10, vZeile));
this.add(lbVeraDatumVon, null);

lbVeraDatumBis.setText("bis");
lbVeraDatumBis.setForeground(Color.black);
lbVeraDatumBis.setBounds(new Rectangle(vZeile*11-10,vZeile*10,
    vSpalte*10,vZeile));
this.add(lbVeraDatumBis, null);

tfVeraDatumVon.setBounds(new Rectangle(vSpalte * 10, vZeile*10,
    vSpalte * 8, vZeile));
tfVeraDatumBis.setBounds(new Rectangle(vSpalte * 24, vZeile*10,
    vSpalte * 8, vZeile));
this.add(tfVeraDatumVon, null);
this.add(tfVeraDatumBis, null);

// JTable für Veranstaltungen erstellen
spVeranstaltungen.setBounds(new Rectangle(vSpalte*35 , vZeile*4,
    vSpalte * 43, vZeile*23));
this.add(spVeranstaltungen, null);

// Ueberschrift einfüegen
lbUeberschrift.setBounds(new Rectangle(20, 0, vSpalte * 55, 50));
lbUeberschrift.setText("OnlineTicket - Veranstaltung auswählen");
lbUeberschrift.setForeground(Color.red);
lbUeberschrift.setFont(new Font("Arial",0,20));
this.add(lbUeberschrift, null);

// Informationen zur Veranstaltung
tpVeranstaltungInfo.setBounds(new Rectangle(20,vZeile*14,
    vSpalte*30,vZeile*15));
spVeranstaltungInfo.setBounds(new Rectangle(20,vZeile*14,
    vSpalte*30,vZeile*15));
tpVeranstaltungInfo.setEnabled(false);
this.add(spVeranstaltungInfo);
spVeranstaltungInfo.getViewPort().add(tpVeranstaltungInfo);

// Buttons erstellen
// Suchen
btSuchen.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        tpVeranstaltungInfo.setText("");
        fuelle_taVeranstaltungen();
    }
});
btSuchen.setLabel("Suchen");
btSuchen.setBounds(new Rectangle(20, vZeile*12, vSpalte * 30, vZeile));
this.add(btSuchen, null);

// Bestellen
btBestellen.setText("Bestellen");
btBestellen.setEnabled(false);
btBestellen.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // aus JTable selektierte Veranstaltung auslesen
        String vVeraIDSelected = (String) taVeranstaltungen.getValueAt(
            taVeranstaltungen.getSelectedRow(),0);
```

```

        vMasterApplet.panelPlatzwahlAktivieren(vVeraIDSelected);
    }
});

btBestellen.setLabel("Bestellen");
btBestellen.setBounds(new Rectangle(vSpalte*35,vZeile*28,
                                   vSpalte* 43, vZeile));
this.add(btBestellen, null);
}

//-----fuelle_taVeranstaltungen-----
private void fuelle_taVeranstaltungen() {
    // Veranstaltungsdaten aus Bean in einen Vektor einlesen
    try {
        Vector ve_veranstaltungen_data = vMasterApplet.remoteTicketEJB.
            getVeranstaltungen(tfVeraBezeichnung.getText(),
                              tfVeraOrt.getText(),
                              tfVeraInterpret.getText(),
                              tfVeraDatumVon.getText(),
                              tfVeraDatumBis.getText());

        // Vektordaten an JTable übergeben
        tmVeranstaltungen.setTableDataRow(ve_veranstaltungen_data);

        // Button Bestellen deaktivieren
        btBestellen.setEnabled(false);

        // Selektion entfernen
        taVeranstaltungen.setRowSelectionInterval(0,0);
        taVeranstaltungen.removeRowSelectionInterval(0,0);
    } catch(Exception e){};
}

private void init_taVeranstaltungen() {
    // initialisiert die JTable für die Anzeige der Veranstaltungen

    // Festlegen der Spaltenüberschriften
    String [] vlabels= {"ID","Bezeichnung","Datum","Ort"};

    // Festlegen der Spaltenanfangswerte
    Object [] vdefault= {"","","",""};

    // Erzeugen des JTableModel, welches die Daten verwaltet
    tmVeranstaltungen = new genTableModel(vlabels,vdefault,0);

    // Erzeugen der JTable
    taVeranstaltungen = new JTable(tmVeranstaltungen);
    taVeranstaltungen.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    // Mouse Listener der JTable hinzufügen
    // um auf Benutzeraktionen innerhalb der JTable reagieren zu können
    taVeranstaltungen.addMouseListener(new java.awt.event.MouseListener() {
        public void mouseClicked(MouseEvent evt) {
            // wenn Benutzer Maus gedrückt hat, Zeile ermitteln
            String vVeraIDSelected = (String) taVeranstaltungen.getValueAt(
                taVeranstaltungen.getSelectedRow(),0);

            // Button Bestellen aktivieren
            btBestellen.setEnabled(true);
            // Informationen über die selektierte Veranstaltung anzeigen
            try{
                DefaultStyledDocument dsd = new DefaultStyledDocument();
                Style ts = tpVeranstaltungInfo.addStyle("standard",

```

```

        tpVeranstaltungInfo.getLogicalStyle());
        StyleConstants.setFontSize(ts,12);
        StyleConstants.setFontFamily(ts,"Arial");
        dsd.insertString(0,vMasterApplet.remoteTicketEJB.
            getVeranstaltungInfo(vVeraIDSelected, "All"),ts);
        tpVeranstaltungInfo.setDocument(dsd);
    }catch(Exception e) {}
}

    public void mouseEntered(MouseEvent evt) {}
    public void mouseExited(MouseEvent evt) {}
    public void mousePressed(MouseEvent evt) {}
    public void mouseReleased(MouseEvent evt) {}
});

    // Scrollpane für die JTable
    spVeranstaltungen = JTable.createScrollPaneForTable( taVeranstaltungen );
}
}
//-----Ende der Klasse PanelVeranstaltungAuswaehlen-----

```

Quellcode der Klasse "PanelPlatzwahl"

```

//=====
// Klasse:      PanelPlatzwahl
// Projekt:      OnlineTicket
// Version:      1.0 vom 23.06.2000
// Autor:        Oliver Hüskes & Michael Zeidler
// Beschreibung: Maske zur Platzauswahl für eine Veranstaltung
//=====

package client;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.util.*;

public class PanelPlatzwahl extends JPanel {

    // Referenz auf das Startapplet AppletTicketShop
    AppletTicketShop vMasterApplet ;

    // Überschrift des Panels
    JLabel lbUeberschrift = new JLabel() ;

    // globale Variablen
    String vVeraID;
    int vBuchID;

    // Konstanten die Zeilen und Spaltengröße definieren
    int vZeile = 20;
    int vSpalte = 10;

    // Komponenten des Panels

```

[illegible]

```
this.add(coAnzahlTicket);

// List Preiskategorie mit Daten aus DB
Vector vePreiskategorien = vMasterApplet.remoteTicketEJB.
    getPreiskategorien(vVeraID);

lbPreiskategorie.setText("Preiskategorie");
lbPreiskategorie.setBounds(new Rectangle(vSpalte * 7, vZeile*3-10,
    vSpalte * 10, vZeile));
lbPreiskategorie.setForeground(Color.black);
this.add(lbPreiskategorie);
coPreiskategorie = new JComboBox(vePreiskategorien);
coPreiskategorie.setBounds(new Rectangle(vSpalte * 7, vZeile*4-10,
    vSpalte * 17+15, vZeile));
this.add(coPreiskategorie);

// JTable mit freien Plätzen
spFreiePlaetze.setBounds(new Rectangle(vSpalte*1 , vZeile*7,
    vSpalte*23+15, vZeile*21-15));
this.add(spFreiePlaetze, null);

// Bild des Veranstaltungsortes
btBild.setBounds(new Rectangle(vSpalte*25+10 , vZeile * 7,
    vSpalte * 53, vZeile*22));

// Bild aus Datenbank laden und anzeigen
byte[] vBildArray = vMasterApplet.remoteTicketEJB.
    getVeranstaltungsortBild(vVeraID);

iVeraOrt= Toolkit.getDefaultToolkit().createImage(vBildArray);
iVeraOrt = iVeraOrt.getScaledInstance(vSpalte * 53,vZeile*22,
    iVeraOrt.SCALE_SMOOTH);
btBild.setIcon(new ImageIcon(iVeraOrt) );
btBild.repaint();
this.add(btBild, null);

// Veranstaltungsinformationen anzeigen
lbVeraInfo.setBounds(new Rectangle(vSpalte*23 , vZeile*3-7,
    vSpalte * 55, vZeile*1));
lbVeraInfo.setText(vMasterApplet.remoteTicketEJB.getVeranstaltungInfo(
    (vVeraID, "Name") ));
lbVeraInfo.setFont(new Font("Arial",0,18));
lbVeraInfo.setForeground(Color.black);
lbVeraInfo.setHorizontalAlignment(SwingConstants.CENTER);
this.add(lbVeraInfo, null);

lbVeraInfo2.setBounds(new Rectangle(vSpalte*23,vZeile *4-10,
    vSpalte*55,vZeile*2));
lbVeraInfo2.setText(vMasterApplet.remoteTicketEJB.getVeranstaltungInfo(
    vVeraID, "Daten") );
lbVeraInfo2.setHorizontalAlignment(SwingConstants.CENTER);
lbVeraInfo2.setForeground(Color.black);
this.add(lbVeraInfo2, null);

// Ueberschrift einfuegen
lbUeberschrift.setBounds(new Rectangle(10, 0, vSpalte * 55, 50));
lbUeberschrift.setText("OnlineTicket - Platzwahl");
lbUeberschrift.setForeground(Color.red);
lbUeberschrift.setFont(new Font("Arial",0,20));
```



```

this.add(lbUeberschrift, null);

// Buttons erstellen
// Veranstaltung suchen
// Action Listener für Button
btSuchen.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        fuelle_taFreiePlaetze();
    }
});
btSuchen.setLabel("Plätze suchen");
btSuchen.setBounds(new Rectangle(vSpalte*1 ,vZeile*5,
                                vSpalte*24+5,vZeile));
this.add(btSuchen, null);

// Plaetze buchen
btOK.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {

        // Benutzerauswahl auslesen und Buchung vornehmen
        String vBlockSelected = (String) taFreiePlaetze.getValueAt(
                                taFreiePlaetze.getSelectedRow(),2);
        String vReiheSelected = (String) taFreiePlaetze.getValueAt(
                                taFreiePlaetze.getSelectedRow(),3);
        String vAnzahlTickets = (String) coAnzahlTicket.getSelectedItemAt();

        try{
            // neue Buchung in Datenbank über Bean anlegen
            vBuchID = vMasterApplet.remoteTicketEJB.insertBuchung(
                                vVeraID, vBlockSelected,
                                vReiheSelected,vAnzahlTickets);
        } catch (Exception ex){};
        // Maske BuchungsdatenErfassen aktivieren
        vMasterApplet.panelBuchungsdatenErfassenAktivieren(vVeraID,
                                (new Integer (vBuchID)).toString());
    }
});
btOK.setLabel("OK");
btOK.setEnabled(false);
btOK.setBounds(new Rectangle(vSpalte*1 , vZeile * 28,
                                vSpalte * 12,vZeile));
this.add(btOK, null);

// zurück zur Startseite
btStartseite.setText("Startseite");
btStartseite.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // zurück zur Startseite
        vMasterApplet.panelPlatzwahlLoeschen();
    }
});
btStartseite.setLabel("Abbrechen");
btStartseite.setBounds(new Rectangle(vSpalte*14 , vZeile * 28,
                                vSpalte * 12-5, vZeile));
this.add(btStartseite, null);

} catch (Exception e){
    System.out.println("Fehler in INIT:"+e);}
}

//-----fuelle_taFreiePlaetze-----

```

```

// anhand der Benutzerauswahl die JTable füllen
private void fuehle_taFreiePlaetze() {
    try {
        // Benutzerauswahl auslesen
        Vector vePreiskategorieSelected =
            (Vector) coPreiskategorie.getSelectedItemAt();

        String vPreiskategorieSelected =
            (String) vePreiskategorieSelected.elementAt(0);

        String vAnzahlTickets = (String) coAnzahlTicket.getSelectedItemAt();
        // Freie Plätze aus Bean in einen Vektor einlesen
        Vector veFreiePlaetze = vMasterApplet.remoteTicketEJB.
            getVerfuegbarePlaetze(vVeraID ,vPreiskategorieSelected,vAnzahlTickets);
        // Vektordaten in JTable übertragen
        tmFreiePlaetze.setTableDataRow(veFreiePlaetze);
        // Button OK deaktivieren
        btOK.setEnabled(false);
        // Selektion in JTable ausschalten
        taFreiePlaetze.setRowSelectionInterval(0,0);
        taFreiePlaetze.removeRowSelectionInterval(0,0);
    } catch (Exception e){};
}
//-----init_taFreiePlaetze-----
// JTable initialisieren
private void init_taFreiePlaetze() {
    // initialisiert die JTable für die Anzeiger der Veranstaltungen
    // Festlegen der Spaltenüberschriften
    String [] vlabels= {"Preiskategorie","Preis","Block","Reihe","frei"};

    // Festlegen der Spaltenanfangswerte
    Object [] vdefault= {"","","","",""};
    // Erzeugen des JTableModel, welches die Daten verwaltet
    tmFreiePlaetze = new genTableModel(vlabels,vdefault,30);
    // Erzeugen der JTable
    taFreiePlaetze = new JTable(tmFreiePlaetze);
    taFreiePlaetze.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
    taFreiePlaetze.addMouseListener(new java.awt.event.MouseListener() {
        public void mouseClicked(MouseEvent evt){
            btOK.setEnabled(true);
        }

        public void mouseEntered(MouseEvent evt) {}
        public void mouseExited(MouseEvent evt) {}
        public void mousePressed(MouseEvent evt) {}
        public void mouseReleased(MouseEvent evt) {}
    });

    // Erzeugen einer Scrollpane für die JTable
    spFreiePlaetze = JTable.createScrollPaneForTable( taFreiePlaetze );
}
}
//-----Ende der Klasse PanelPlatzwahl-----

```

Quellcode der Klasse " PanelBuchungsdatenErfassen "

```
//=====
// Klasse:      PanelBuchungsdatenErfassen
// Projekt:     OnlineTicket
// Version:     1.0 vom 23.06.2000
// Autor:      Oliver Hüskes & Michael Zeidler
// Beschreibung: Maske zum Erfassen der Buchungsdaten
//              Hier kann sich ein bestehender Kunde einloggen
//              oder ein neuer Kunde angelegt werden
//=====
package client;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;
import java.util.*;

public class PanelBuchungsdatenErfassen extends JPanel {

    // Referenz auf das Startapplet AppletTicketShop
    AppletTicketShop vMasterApplet ;

    // Überschrift des Panels
    JLabel lbUeberschrift    = new JLabel();

    // globale Variablen
    String  vVeraID;
    String  vBuchID ;
    Vector  veKundendaten;

    // Konstanten die Zeilen und Spaltengröße definieren
    int vZeile  = 20;
    int vSpalte = 10;

    // Komponenten des Panels
    // Kunde-Nr.
    JTextField tfKundeNr  = new JTextField();
    JLabel lbKundeNr      = new JLabel();

    // Kunde-Passwort
    JPasswordField tfKundePWD = new JPasswordField();
    JLabel lbKundePWD        = new JLabel();

    // Kunde-Name
    JTextField tfKundeName  = new JTextField();
    JLabel lbKundeName      = new JLabel();

    // Kunde-Strasse
    JTextField tfKundeStrasse = new JTextField();
    JLabel lbKundeStrasse     = new JLabel();

    // Kunde-PLZ
    JTextField tfKundePLZ   = new JTextField();
    JLabel lbKundePLZ       = new JLabel();

    // Kunde-Ort
    JTextField tfKundeOrt   = new JTextField();
    JLabel lbKundeOrt       = new JLabel();
}
```

```

// Kunde-Email
JTextField tfKundeEmail = new JTextField();
JLabel lbKundeEmail = new JLabel();

// Zahlungsart
JLabel lbZahlungsart = new JLabel();
JComboBox coZahlungsart ;

// Kunde-Kreditkarte-Nr
JTextField tfKundeKreditkarteNr = new JTextField();
JLabel lbKundeKreditkarteNr = new JLabel();

// Kunde-Kreditkarte-Gueltig-Bis
JTextField tfKundeKreditkarteGueltigBis = new JTextField();
JLabel lbKundeKreditkarteGueltigBis = new JLabel();

// Information zur Buchung
JTextPane tpBuchungInfo = new JTextPane();
JScrollPane spBuchungInfo = new JScrollPane() ;

// Buttons
JButton btSpeichern = new JButton();
JButton btAbbrechen = new JButton();
JButton btLogin = new JButton();
JButton btNeuerKunde = new JButton();
JButton btReset = new JButton();

//-----Konstruktor-----
public PanelBuchungsdatenErfassen(AppletTicketShop p_masterApplet,
                                   String pVeraID,
                                   String pBuchID ) {

    // Referenz des MasterApplets speichern
    vMasterApplet = p_masterApplet;
    vBuchID = pBuchID ;
    vVeraID = pVeraID ;

    // Panel initialisieren
    jbInit();
}

//-----Initialisierung des Panel-----
private void jbInit() {
try{
    // Grösse des Panels bestimmen
    this.setSize(800,600);
    // Keinen Layoutmanager benutzen
    this.setLayout(null);

    // Kunde-Nr.
    tfKundeNr.setBounds(new Rectangle(vSpalte*12,vZeile*4,
                                       vSpalte*17,vZeile));
    this.add(tfKundeNr, null);
    lbKundeNr.setText("Kunden-Nr.");
    lbKundeNr.setBounds(new Rectangle(20, vZeile*4, vSpalte * 7, vZeile));
    lbKundeNr.setForeground(Color.black);
    this.add(lbKundeNr);
    // Passwort

```

```
tfKundePWD.setBounds(new Rectangle(vSpalte*12, vZeile*6,
                                   vSpalte*17,vZeile));
this.add(tfKundePWD, null);
lbKundePWD.setText("Passwort");
lbKundePWD.setBounds(new Rectangle(20, vZeile*6, vSpalte * 10, vZeile));
lbKundePWD.setForeground(Color.black);
this.add(lbKundePWD);
// Kunde-Name
tfKundeName.setBounds(new Rectangle(vSpalte*12,vZeile*8,
                                   vSpalte*17,vZeile));
this.add(tfKundeName, null);
lbKundeName.setText("Nach-, Vorname");
lbKundeName.setBounds(new Rectangle(20, vZeile*8,
                                   vSpalte * 10, vZeile));
lbKundeName.setForeground(Color.black);
this.add(lbKundeName);
// Kunde-Strasse
tfKundeStrasse.setBounds(new Rectangle(vSpalte*12,vZeile*10,
                                       vSpalte*17,vZeile));
this.add(tfKundeStrasse, null);
lbKundeStrasse.setText("Straße");
lbKundeStrasse.setBounds(new Rectangle(20, vZeile*10,
                                       vSpalte * 10, vZeile));
lbKundeStrasse.setForeground(Color.black);
this.add(lbKundeStrasse);
// Kunde-PLZ
tfKundePLZ.setBounds(new Rectangle(vSpalte*12,vZeile*12,
                                   vSpalte* 5,vZeile));
this.add(tfKundePLZ, null);
lbKundePLZ.setText("PLZ");
lbKundePLZ.setBounds(new Rectangle(20, vZeile*12,
                                   vSpalte * 10, vZeile));
lbKundePLZ.setForeground(Color.black);
this.add(lbKundePLZ);
// Kunde-Ort
tfKundeOrt.setBounds(new Rectangle(vSpalte*12,vZeile*14,
                                   vSpalte*17,vZeile));
this.add(tfKundeOrt, null);
lbKundeOrt.setText("Ort");
lbKundeOrt.setBounds(new Rectangle(20,vZeile*14,
                                   vSpalte * 10, vZeile));
lbKundeOrt.setForeground(Color.black);
this.add(lbKundeOrt);
// Kunde Email
tfKundeEmail.setBounds(new Rectangle(vSpalte*12,vZeile*16,
                                    vSpalte*17,vZeile));
this.add(tfKundeEmail, null);
lbKundeEmail.setText("E-Mail");
lbKundeEmail.setBounds(new Rectangle(20, vZeile*16,
                                    vSpalte * 10, vZeile));
lbKundeEmail.setForeground(Color.black);
this.add(lbKundeEmail);

// Zahlungsart
String vValues[] = {"VisaCard","EuroCard","AmericanExpress"};
coZahlungsart = new JComboBox(vValues);
coZahlungsart.setBounds(new Rectangle(vSpalte * 12, vZeile*18,
                                     vSpalte * 17, vZeile));
this.add(coZahlungsart);
lbZahlungsart.setText("Zahlungsart");
lbZahlungsart.setBounds(new Rectangle(20, vZeile*18,
```

```

                                vSpalte * 10, vZeile));
lbZahlungsart.setForeground(Color.black);
this.add(lbZahlungsart);
// Kreditkarte Nr
tfKundeKreditkarteNr.setBounds(new Rectangle(vSpalte * 12, vZeile*20,
                                                vSpalte * 17, vZeile));

this.add(tfKundeKreditkarteNr, null);
lbKundeKreditkarteNr.setText("Karten-Nr.");
lbKundeKreditkarteNr.setBounds(new Rectangle(20,vZeile*20,
                                                vSpalte*10,vZeile));
lbKundeKreditkarteNr.setForeground(Color.black);
this.add(lbKundeKreditkarteNr);
// Kreditkarte Gueltig Bis
tfKundeKreditkarteGueltigBis.setBounds(
    new Rectangle(vSpalte * 12,vZeile*22,vSpalte * 17, vZeile));
this.add(tfKundeKreditkarteGueltigBis, null);
lbKundeKreditkarteGueltigBis.setText("Karte gueltig bis");
lbKundeKreditkarteGueltigBis.setBounds(
    new Rectangle(20, vZeile*22, vSpalte * 10, vZeile));
lbKundeKreditkarteGueltigBis.setForeground(Color.black);
this.add(lbKundeKreditkarteGueltigBis);
// Kunden-Felder ausblenden
showKundendaten(false);
// Reset-Button ausblenden
btReset.setVisible(false);

// Ueberschrift einfuegen
lbUeberschrift.setBounds(new Rectangle(20, 0, vSpalte * 55, 50));
lbUeberschrift.setText("OnlineTicket - Buchungsdaten erfassen");
lbUeberschrift.setForeground(Color.red);
lbUeberschrift.setFont(new Font("Arial",0,20));
this.add(lbUeberschrift, null);

// Informationen zur Buchung in Form einer TextPane darstellen
tpBuchungInfo.setBounds(new Rectangle(vSpalte* 30, vZeile*4,
                                       vSpalte * 48, vZeile*25));
spBuchungInfo.setBounds(new Rectangle(vSpalte* 30, vZeile*4,
                                       vSpalte * 48, vZeile*25));
tpBuchungInfo.setEnabled(false);
this.add(spBuchungInfo);
spBuchungInfo.getViewport().add(tpBuchungInfo);
// formatierten Text in die TextPane hinzufuegen

DefaultStyledDocument dsd = new DefaultStyledDocument();
Style tsStandard = tpBuchungInfo.addStyle("standard",
                                           tpBuchungInfo.getLogicalStyle());

// Schriftart definieren
StyleConstants.setFontSize(tsStandard,12);
StyleConstants.setFontFamily(tsStandard,"Courier");
// Text bei der Bean anfordern
dsd.insertString(0,vMasterApplet.remoteTicketEJB.
                getBuchungInfo(vBuchID,vVeraID, "Screen"),tsStandard);

tpBuchungInfo.setDocument(dsd);

// Buttons erstellen
// Veranstaltung buchen
btSpeichern.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
    }
});

```

```

btSpeichern.setLabel("Buchen");
btSpeichern.setBounds(new Rectangle(20,vZeile*28,vSpalte* 13,vZeile));
this.add(btSpeichern, null);
// Action Listener für Button
btSpeichern.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try{
            // Überprüfen ob ein Kunde neu angelegt worden ist
            if (tfKundeNr.getText().equals("wird automatisch vergeben")){
                // Benutzerauswahl auslesen und Buchung vornehmen
                String vZahlungsart= (String) coZahlungsart.getSelectedItem();

                // neuen Kundendatensatz anlegen und Kunden-ID anzeigen
                String vKundID = vMasterApplet.remoteTicketEJB.insertNeuenKunde
                    (tfKundeName.getText(),
                     tfKundeStrasse.getText(),
                     tfKundePLZ.getText(),
                     tfKundeOrt.getText(),
                     tfKundeEmail.getText(),
                     (String) coZahlungsart.getSelectedItem(),
                     tfKundeKreditkarteNr.getText(),
                     tfKundeKreditkarteGueltigBis.getText(),
                     tfKundePWD.getText());

                // Kunden-ID in Maske anzeigen
                tfKundeNr.setText(vKundID);
            }
            // Wenn Kunde Daten geändert hat
            else if (changedKundendaten()){
                // Kundendatensatz ändern
                String vKundID = vMasterApplet.remoteTicketEJB.updateKunde
                    (tfKundeNr.getText(),
                     tfKundeName.getText(),
                     tfKundeStrasse.getText(),
                     tfKundePLZ.getText(),
                     tfKundeOrt.getText(),
                     tfKundeEmail.getText(),
                     (String) coZahlungsart.getSelectedItem(),
                     tfKundeKreditkarteNr.getText(),
                     tfKundeKreditkarteGueltigBis.getText(),
                     tfKundePWD.getText());
            }

            // Kunde zu Buchung zuordnen
            int vReturn = vMasterApplet.remoteTicketEJB.ordneBuchungZuKunde
                (tfKundeNr.getText(), vBuchID);
            MessageDialog.createMessageDialog("Buchung wurde erfolgreich "+
                "durchgeführt.\nDie Tickets werden Ihnen umgehend "+
                "zugesendet.\nIhre Buchungsnummer lautet "+vBuchID+
                ".\n\nVielen Dank! Ihr Ticket-Shop-Team.", "OK", "");

            // wenn eine E-Mail-Adresse existiert, wird dem Kunden eine
            // Buchungsbestätigung zugesendet
            if (!tfKundeEmail.getText().equals(""))
                // E-Mail versenden
                vMasterApplet.remoteTicketEJB.sendEmail("service@OnlineTicket.de",
                    tfKundeEmail.getText(), "Buchungsbestätigung OnlineTicket",
                    vMasterApplet.remoteTicketEJB.getBuchungInfo(vBuchID,vVeraID,
                        "Email"));
            // Maske BuchungsdatenErfassen schließen
            vMasterApplet.panelBuchungsdatenErfassenLoeschen();
        } catch (Exception ex){};
    }
}

```

```
});

// Eingabe abbrechen und zur Startseite zurückkehren
btAbbrechen.setLabel("Abbrechen");
btAbbrechen.setBounds(new Rectangle(vSpalte*16,vZeile*28,
                                     vSpalte*13,vZeile));

this.add(btAbbrechen, null);
// Action Listener für Button
btAbbrechen.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try{
            // angelegte Reservierung (Buchung ohne Kundenbezug) löschen
            vMasterApplet.remoteTicketEJB.deleteBuchung(vBuchID);
            // Kundendatenmaske loeschen und zur Startseite zurückkehren
            vMasterApplet.panelBuchungsdatenErfassenLoeschen();
        } catch (Exception ex){};
    }
});

// bestehenden Kunden einloggen
btLogin.setLabel("Login");
btLogin.setBounds(new Rectangle(20 , vZeile * 8, vSpalte * 12, vZeile));
this.add(btLogin, null);
// Action Listener für Button
btLogin.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        try{
            // Kundendaten holen
            veKundendaten = vMasterApplet.remoteTicketEJB.getKundendaten
                           (tfKundeNr.getText(), tfKundePWD.getText());
        } catch (Exception ex){};
        // Wenn Login erfolgreich Daten anzeige
        if (veKundendaten.size() > 0){
            showKundendaten(true);
            btNeuerKunde.setVisible(false);
            btLogin.setVisible(false);
            btReset.setVisible(true);
            tfKundeName.setText((String) veKundendaten.elementAt(0));
            tfKundeOrt.setText((String) veKundendaten.elementAt(1));
            tfKundeEmail.setText((String) veKundendaten.elementAt(2));
            tfKundePLZ.setText((String) veKundendaten.elementAt(3));
            tfKundeStrasse.setText((String) veKundendaten.elementAt(4));
            coZahlungsart.setSelectedItem((String) veKundendaten.elementAt(5));
            tfKundeKreditkarteNr.setText((String) veKundendaten.elementAt(6));
            tfKundeKreditkarteGueltigBis.setText((String)
                                                veKundendaten.elementAt(7));
        }
        // wenn keine Daten gefunden werden, Fehlermeldung ausgeben
        else
            MessageDialog.createMessageDialog("Fehler beim Einloggen.\nBitte "+
                                              "überprüfen Sie Kunden-Nr. und "+
                                              "Passwort! ", "OK", "");
    }
});

// neuen Kunden anlegen
btNeuerKunde.setLabel("Neuer Kunde");
btNeuerKunde.setBounds(new Rectangle(vSpalte*17 , vZeile * 8,
                                     vSpalte * 12, vZeile));

this.add(btNeuerKunde, null);
// Action Listener für Button
```



```

        btNeuerKunde.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showKundendaten(true);
                btNeuerKunde.setVisible(false);
                btLogin.setVisible(false);
                btReset.setVisible(true);
                initKundendaten();
                tfKundeNr.setText("wird automatisch vergeben");
            }
        });

        // Reset
        btReset.setLabel("Reset");
        btReset.setBounds(new Rectangle(20,vZeile * 24, vSpalte * 27, vZeile));
        this.add(btReset, null);
        // Action Listener für Button
        btReset.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(ActionEvent e) {
                showKundendaten(false);
                btNeuerKunde.setVisible(true);
                btLogin.setVisible(true);
                btReset.setVisible(false);
                initKundendaten();
            }
        });
    }
    catch(Exception e) {}
}

//-----showKundendaten-----
// blendet die Kunden-Eingabefelder ein/aus
public void showKundendaten(boolean pModus){

    tfKundeName.setVisible(pModus);
    tfKundeStrasse.setVisible(pModus);
    tfKundeOrt.setVisible(pModus);
    tfKundeEmail.setVisible(pModus);
    tfKundePLZ.setVisible(pModus);
    coZahlungsart.setVisible(pModus);
    tfKundeKreditkarteNr.setVisible(pModus);
    tfKundeKreditkarteGueltigBis.setVisible(pModus);

    lbKundeName.setVisible(pModus);
    lbKundeName.setVisible(pModus);
    lbKundeStrasse.setVisible(pModus);
    lbKundeOrt.setVisible(pModus);
    lbKundeEmail.setVisible(pModus);
    lbKundePLZ.setVisible(pModus);
    lbZahlungsart.setVisible(pModus);
    lbKundeKreditkarteNr.setVisible(pModus);
    lbKundeKreditkarteGueltigBis.setVisible(pModus);
    tfKundeNr.setEnabled(!pModus);
    btSpeichern.setEnabled(pModus);
}

//-----initKundendaten-----
// initialisiert die Kunden-Eingabefelder

public void initKundendaten(){
    tfKundeNr.setText("");
    tfKundePWD.setText("");
    tfKundeName.setText("");

```

```

        tfKundeStrasse.setText("");
        tfKundeOrt.setText("");
        tfKundePLZ.setText("");
        coZahlungsart.setSelectedIndex(0);
        tfKundeKreditkarteNr.setText("");
        tfKundeKreditkarteGueltigBis.setText("");
    }

//-----changedKundendaten-----
// überprüft ob aktuellen Kundendaten am Bildschirm identisch sind mit den
// Daten, die aus der Datenbank geladen wurden

public boolean changedKundendaten() {
    Vector veKundendatenAkt = new Vector();

    veKundendatenAkt.addElement((String)tfKundeName.getText());
    veKundendatenAkt.addElement((String)tfKundeOrt.getText());
    veKundendatenAkt.addElement((String)tfKundeEmail.getText());
    veKundendatenAkt.addElement((String)tfKundePLZ.getText());
    veKundendatenAkt.addElement((String)tfKundeStrasse.getText());
    veKundendatenAkt.addElement((String)coZahlungsart.getSelectedItem());
    veKundendatenAkt.addElement((String)tfKundeKreditkarteNr.getText());
    veKundendatenAkt.addElement(
        (String)tfKundeKreditkarteGueltigBis.getText());
    veKundendatenAkt.addElement((String)tfKundePWD.getText());

    return(!veKundendatenAkt.equals( veKundendaten));
}
}
//-----Ende der Klasse PanelBuchungsdatenErfassen-----

```

Quellcode der Klasse "genTableModel"

```

//=====
// Klasse:      genTableModel
// Projekt:      OnlineTicket
// Version:      1.0 vom 23.06.2000
// Autor:        Oliver Hüskes & Michael Zeidler
// Beschreibung: Klasse um das TableModel einer JTable manipulieren zu
//              können: z.B.Daten in Form eines Vectors einfügen
//=====

package client;

import java.util.*;
import javax.swing.table.*;

class genTableModel extends AbstractTableModel {

    Vector veData;                // Tabellen-Daten
    String[] vColumnNames;        // Spalten-Namen.

    //-----Konstruktor-----
    // Initialisiert die Tabellenstruktur, und füllt die Tabelle
    // mit den übergebenen Default-Werten

```

```
// Parameter: pColumns: array mit den Spaltenüberschriften
//              pDefaultv: array mit den Defaultwerten
//              pRows:      Anzahl der Tabellenspalten
public genTableModel(String pColumns[], Object pDefaultv[], int pRows) {

    // Initialisiert Anzahl der Spalten und Spaltenüberschriften
    vColumnNames = new String[pColumns.length];
    for (int i=0; i<pColumns.length; i++)
        vColumnNames[i] = new String(pColumns[i]);

    // Erzeugen eines Daten-Vektors und füllen mit Default-Werten
    veData = new Vector();
    for (int i=0; i<pRows; i++) {
        Vector veCols = new Vector();
        for (int j=0; j<pColumns.length; j++)
            veCols.addElement(pDefaultv[j]);
        veData.addElement(veCols);
    }
}

//----- Anzahl der Spalten ermitteln-----
public int getColumnCount() {
    return vColumnNames.length;
}

//----- Anzahl der Zeilen ermitteln-----
public int getRowCount() {
    return veData.size();
}

//----- Spalten-Name ermitteln-----
// Parameter: pCol: Nummer der Spalte
public String getColumnName(int pCol){
    return vColumnNames[pCol];
}

//----- Zellen-Wert ermitteln-----
// Parameter: pCol: Nummer der Spalte
//              pRow: Nummer der Zeile
public Object getValueAt(int pRow, int pCol) {
    Vector veColvector = (Vector) veData.elementAt(pRow);
    return veColvector.elementAt(pCol);
}

//----- Spalten-Klasse ermitteln-----
// Parameter: pCol: Nummer der Spalte
public Class getColumnClass(int pCol) {
    return getValueAt(0,pCol).getClass();
}

//----- Zellen-Wert setzen -----
// Parameter: pObj: Wert
//              pCol: Nummer der Spalte
//              pRow: Nummer der Zeile
public void setValueAt( Object pObj, int pRow, int pCol) {
    Vector veColvector = (Vector) veData.elementAt(pRow);
```

```

        veColvector.setElementAt(pObj, pCol);
    }

    //----- Tabellen-Daten setzen -----
    // Parameter: pData: Tabellen-Daten
    public void setTableDataRow(Vector pData) {
        veData = pData;
        super.fireTableDataChanged();
    }
}
//-----Ende der Klasse genTableModel-----

```

Quellcode der Klasse "MessageDialog"

```

//=====
// Klasse:      MessageDialog
// Projekt:      OnlineTicket
// Version:      1.0 vom 23.06.2000
// Autor:        Oliver Hüskes & Michael Zeidler
// Beschreibung: Klasse zur Erzeugung eines Meldungsfenster
//=====

package client;
import java.awt.*;
import javax.swing.*;

public class MessageDialog extends Dialog {
    // Instanzvariablen des Message-Dialogs
    protected Button button1;
    protected Button button2;
    protected static Frame createdFrame;
    static int buttonPressed = 0;

    //-----Konstruktor-----
    // Initialisiert die Tabellenstruktur, und füllt die Tabelle
    // mit den übergebenen Default-Werten
    // Parameter: parent:
    //             message:      auszugebende Nachricht
    //             labelButton1: Beschriftung des 1.Button
    //             labelButton2: Beschriftung des 2.Button

    public MessageDialog (Frame parent,
                          String message,
                          String labelButton1,
                          String labelButton2) {

        // Erzeugen der Dialog Komponenten
        super(parent,true);
        button1 = new Button(labelButton1);
        button2 = new Button(labelButton2);
        JTextArea messageLabel = new JTextArea(message);

        messageLabel.setOpaque(true);
        messageLabel.setBackground(Color.lightGray);
        add(messageLabel);
        add(button1);
    }
}

```

```
        if (! labelButton2.equals("")) {
            add(button2);
        }

        // Als LayoutManager FlowLayout auswählen
        setLayout(new FlowLayout());

        // Komponenten neu ausrichten
        pack();
    }

    //-----Action Methode-----
    // Wenn Benutzer einen der beiden Buttons drückt, wird diese Methode
    // ausgeführt. Der gedrückte Button wird in der variable buttonPressed
    // notiert und danach das Fenster geschlossen
    public boolean action(Event evt, Object whichAction) {
        if (evt.target == button1) {
            buttonPressed = 1;
            hide();
        }
        else
            if (evt.target == button2) {
                buttonPressed = 2;
                hide();
            }

        return true;
    }

    //----- ermittelt den gedrückten Button -----
    public int getButtonPressed() {
        return( buttonPressed);
    }

    //----- Message Dialog erstellen -----
    public static int createMessageDialog(String dialogString,
                                         String labelButton1,
                                         String labelButton2) {
        createdFrame = new Frame("Dialog");
        MessageDialog errorDialog = new MessageDialog
            (createdFrame, dialogString, labelButton1, labelButton2);

        createdFrame.resize(errorDialog.size().
                            width, errorDialog.size().height);
        errorDialog.show();
        return buttonPressed;
    }
}
//-----Ende der Klasse MessageDialog-----
```

Quellcode der Enterprise JavaBean

Quellcode des Home-Interface "TicketHome"

```
//=====
// Interface:      TicketHome (Bean-HomeInterface)
// Projekt:        OnlineTicket
// Version:        1.0 vom 23.06.2000
// Autor:          Oliver Hüskes & Michael Zeidler
// Beschreibung:
//=====

package packagel;

import java.rmi.*;
import javax.ejb.*;

public interface TicketHome extends EJBHome {

    TicketRemote create() throws javax.ejb.CreateException,
    java.rmi.RemoteException;
}

//-----Ende des Home-Interfaces TicketHome-----
```

Quellcode des Remote-Interface "TicketRemote"

```
//=====
// Interface:      TicketRemote (Bean-RemoteInterface)
// Projekt:        OnlineTicket
// Version:        1.0 vom 23.06.2000
// Autor:          Oliver Hüskes & Michael Zeidler
// Beschreibung:
//=====

package oneEJBTicket;

import java.rmi.*;
import javax.ejb.*;

public interface TicketRemote extends EJBObject {

    public int deleteBuchung(java.lang.String pBuchID)
        throws java.rmi.RemoteException;

    public java.lang.String getBuchungInfo(java.lang.String pBuchID,
        java.lang.String pVeraID,
        java.lang.String pModus)
        throws java.rmi.RemoteException;

    public java.util.Vector getKundendaten(java.lang.String pKundID,
        java.lang.String pKundPWD)
```

```
        throws java.rmi.RemoteException;

public java.util.Vector getPreiskategorien(java.lang.String pVeraID)
        throws java.rmi.RemoteException;

public byte[] getVeranstaltungsortBild(java.lang.String pVeorId)
        throws java.rmi.RemoteException;

public java.lang.String getVeranstaltungInfo(java.lang.String pVeraID,
        java.lang.String pModus)
        throws java.rmi.RemoteException;

public java.util.Vector getVeranstaltungen(
        java.lang.String pVeraBezeichnung,
        java.lang.String pVeraOrt,
        java.lang.String pVeraInterpret,
        java.lang.String pVeraDatumVon,
        java.lang.String pVeraDatumBis)
        throws java.rmi.RemoteException;

public java.util.Vector getVerfuegbarePlaetze(
        java.lang.String pVera_ID,
        java.lang.String pPrka_ID,
        java.lang.String pTicketAmzahl)
        throws java.rmi.RemoteException;

public int insertBuchung(java.lang.String pVeraID,
        java.lang.String pBlock,
        java.lang.String pReihe,
        java.lang.String pTicketAnzahl)
        throws java.rmi.RemoteException;

public java.lang.String insertNeuenKunde(
        java.lang.String pKund_Nachame,
        java.lang.String pKund_Strasse,
        java.lang.String pKund_PLZ,
        java.lang.String pKund_Ort,
        java.lang.String pKund_Email,
        java.lang.String pKund_Kreditkarte_Typ,
        java.lang.String pKund_Kreditkarte_Nr,
        java.lang.String pKund_Kreditkarte_Guelting_Bis,
        java.lang.String pKund_PWD)
        throws java.rmi.RemoteException;

public int ordneBuchungZuKunde(java.lang.String pKund_ID,
        java.lang.String pBuch_ID)
        throws java.rmi.RemoteException;

public void sendEmail(java.lang.String vAdressAutor,
        java.lang.String vAdressEmpfaenger,
        java.lang.String vBetreff,
        java.lang.String vText)
        throws java.rmi.RemoteException;

public java.lang.String updateKunde(java.lang.String pKund_Nr,
        java.lang.String pkund_name,
        java.lang.String pKund_Strasse,
        java.lang.String pKund_PLZ,
        java.lang.String pKund_Ort,
        java.lang.String pKund_Email,
        java.lang.String pKund_Kreditkarte_Typ,
        java.lang.String pKund_Kreditkarte_Nr,
```

```

        java.lang.String pKund_Kreditkarte_Gueltig_Bis,
        java.lang.String pKund_PWD)
        throws java.rmi.RemoteException;
    }
//-----Ende des Remote-Interfaces TicketRemote-----

```

Quellcode der Bean-Klasse "Ticket"

```

//=====
// Klasse:      Ticket (Bean-Klasse)
// Projekt:      OnlineTicket
// Version:      1.0 vom 23.06.2000
// Autor:        Oliver Hüskes & Michael Zeidler
// Beschreibung: Enterprise Java Bean, welche die Anwendungslogik beinhaltet
//=====

package oneEJBTicket;

import java.rmi.RemoteException;
import javax.ejb.*;
import java.util.Vector;
import java.sql.*;
import java.text.NumberFormat;
import java.io.*;
import oracle.sql.*;
import oracle.jdbc.driver.*;
import java.util.Locale;

public class Ticket implements SessionBean{

    //-----Konstruktor-----
    public Ticket() {
        try{
            ejbCreate();
        } catch (Exception e){}
    }
    //
    Connection vConnection;

    //-----ejbCreate()-----
    // Methode, die ausgeführt wird wenn eine neue EJB-Instanz erzeugt wurde
    // Hier wird diese Methode verwendet um eine Datenbankverbindung
    // herzustellen
    // Diese Datenbankverbindung kann für alle Datenbankabfragen innerhalb der
    // EJB-Instanz verwendet werden
    public void ejbCreate() throws RemoteException, CreateException {

        try{
            // Oracle JDBC Treiber laden bzw. dem Driver Manager bekannt machen
            Class.forName("oracle.jdbc.driver.OracleDriver");

            // Verbindung zur Datenbank herstellen
            vConnection = DriverManager.getConnection (
                "jdbc:oracle:thin:mze/mze@192.168.5.179:1521:PEAC");

```



```

    } catch (Exception e){
        System.out.println("Fehler beim EJBCREATE:"+e);
    }
}

//-----ejbActivate()-----
public void ejbActivate() throws RemoteException {
}

//-----ejbPassivate()-----
public void ejbPassivate() throws RemoteException {
}

//-----ejbRemove()-----
// Methode, die ausgeführt wird wenn eine EJB-Instanz gelöscht wurde
// Hier wird die Methode verwendet um die Datenbankverbindung zu schließen
public void ejbRemove() throws RemoteException {
    try{
        // Datenbankverbindung schließen
        vConnection.close();
    }catch (Exception e){
        System.out.println("Fehler beim EJBREMOVE:"+e);
    }
}

//-----setSessionContext-----
public void setSessionContext(SessionContext ctx) throws RemoteException {
}

//-----getVeranstaltungen-----
// sucht in der Datenbank nach Veranstaltungen, die den
// Selektionsbedingungen entsprechen und gibt sie als Vektor zurück

public Vector getVeranstaltungen(String pVeraBezeichnung,
                                String pVeraOrt,
                                String pVeraInterpret,
                                String pVeraDatumVon,
                                String pVeraDatumBis)
                                throws java.rmi.RemoteException {

    // Rückgabvektor
    Vector v_zeilen = new Vector();

    // Where Bedingung zusammenstellen
    String v_where = " ";
    if (!(pVeraBezeichnung.equals(""))){
        v_where = v_where+" and vera_bezeichnung like '"+pVeraBezeichnung+"' ";
    }
    if (!(pVeraOrt.equals(""))){
        v_where = v_where + " and veor_bezeichnung like '"+pVeraOrt+"' ";
    }
    if (!(pVeraInterpret.equals(""))){
        v_where = v_where + " and Vera_Interpret like '"+pVeraInterpret+"' ";
    }

    if (!(pVeraDatumVon.equals(""))
        if (!(pVeraDatumBis.equals(""))
            v_where = v_where + " and Vera_Datum between to_date('" +
                pVeraDatumVon+
                "','DD.MM.YYYY') and to_date('" +pVeraDatumBis+"','DD.MM.YYYY')";
        )
    )
}

```



```

        "preis_kategorien,preise where plae_id not "+
        "in ( select  tick_plae_id from tickets) and"+
        " plae_prka_id = prka_id and prka_id = "+
        "prei_prka_id and prei_vera_id = "+pVera_ID;

// Wenn der Benutzer nicht alle Preiskategorien sehen will
if (!pPrka_ID.equals("0"))
    vSelectStatement = vSelectStatement+" and prei_prka_id = "+pPrka_ID;

vSelectStatement = vSelectStatement+ " group by  prei_preis,"+
                                     "prka_bezeichnung,plae_block,"+
                                     "plae_reihe ";

// Anzahl der gewünschten Tickets berücksichtigen
vSelectStatement =
    vSelectStatement+" having Count(*) >= "+pTicketAmzahl;

// Abfrage ausführen
ResultSet v_resultset = v_statement.executeQuery(vSelectStatement);

// Ergebnismenge in Vector umwandeln
while ( v_resultset.next() ) {
    Vector vSpalten = new Vector();
    vSpalten.addElement(v_resultset.getString(1));
    vSpalten.addElement(v_resultset.getString(2));
    vSpalten.addElement(v_resultset.getString(3));
    vSpalten.addElement(v_resultset.getString(4));
    vSpalten.addElement(v_resultset.getString(5));
    vZeilen.addElement(vSpalten);
}

// Statement schließen
v_statement.close();

} catch (Exception e){
    System.out.println("Fehler bei getVerfuegbarePlaetze:"+e);
}

// Vector zurückgeben
return vZeilen;
}

//-----getPreiskategorien-----
// sucht für die übergebene Veranstaltung die vorhandenen Preiskategorien
// und gibt sie als Vektor zurück

public Vector getPreiskategorien(String pVeraID)
    throws java.rmi.RemoteException{
    // Rückgabevektor
    Vector vZeilen = new Vector();

    // erste Zeile in Vektor fest eintragen mit Eintrag "0,
    // alle Preiskategorien
    Vector vSpalten1 = new Vector();
    vSpalten1.addElement("0");
    vSpalten1.addElement("alle");
    vSpalten1.addElement("");
    vZeilen.addElement(vSpalten1);

    try{
        // Statement erstellen

```

```

Statement v_statement = vConnection.createStatement();

// Select ausführen
ResultSet v_resultset =
    v_statement.executeQuery("Select to_CHAR(PRKA_ID) "
        +",PRKA_bezeichnung,PREI_PREIS||' DM' "+
        "from preise,preis_kategorien where prka_ID = "+
        "prei_prka_id and prei_vera_id = "+pVeraID);

// Ergebnismenge in Vektor umwandeln
while ( v_resultset.next() ) {
    Vector vSpalten = new Vector();
    vSpalten.addElement(v_resultset.getString(1));
    vSpalten.addElement(v_resultset.getString(2));
    vSpalten.addElement(v_resultset.getString(3));
    vZeilen.addElement(vSpalten);
}

// Statement schließen
v_statement.close();

} catch (Exception e){
    System.out.println("Fehler bei getPreiskategorien:"+e);
}

// Vektor zurückgeben
return vZeilen;
}

//-----getVeranstaltungInfo-----
// liefert für eine Veranstaltung Informationen in Form eines
// Strings zurück
// Dabei kann die Art der Information durch den Parameter pModus bestimmt
// werden
// pModus = "All"    -> Bezeichnung, Ort,Datum und Beschreibung
//          = "Name"  -> Bezeichnung
//          = "Daten" -> Bezeichnung, Ort und Datum
public String getVeranstaltungInfo(String pVeraID, String pModus)
    throws java.rmi.RemoteException {

    String vVeraBez = "";
    String vVeraBez1 = "";
    String vVeraBez2 = "";
    String vVeraBez3 = "";
    String vVeraBez4 = "";
    String vVeraBez5 = "";
    try{
        // Statement erstellen
        Statement v_statement = vConnection.createStatement();

        // Select ausführen
        ResultSet v_resultset =
            v_statement.executeQuery("select Vera_bezeichnung"+
                ",to_char(vera_datum,'DD.MM.YYYY'),rtrim(vera_beginn)"+
                "||' Uhr',veor_bezeichnung,vera_beschreibung from "+
                "veranstaltungen,veranstaltung_orte where vera_veor_id"
                +" = veor_id and vera_id = "+pVeraID);

        // Ergebnismenge in lokale Variablen schreiben
        while (v_resultset.next() ) {
            vVeraBez1 = v_resultset.getString(1);
            vVeraBez2 = v_resultset.getString(2);

```

```

        vVeraBez3 = v_resultset.getString(3);
        vVeraBez4 = v_resultset.getString(4);
        vVeraBez5 = v_resultset.getString(5);
    }
    // Statement schließen
    v_statement.close();

    // Informationen erstellen
    if (pModus.equals("Name"))
        vVeraBez = vVeraBez1;
    if (pModus.equals("Daten"))
        vVeraBez = "Datum: "+vVeraBez2+"      Ort: "+vVeraBez4+"      "+
            "Beginn: "+vVeraBez3 ;
    if (pModus.equals("All"))
        vVeraBez = vVeraBez1+ " \nDatum: "+vVeraBez2+" \nOrt: "+vVeraBez4+
            " \nBeginn: "+vVeraBez3+" \n\n"+ vVeraBez5;
    } catch (Exception e){
        System.out.println("Fehler beim JDBC:"+e);
    }
    // Information zurückgeben
    return (vVeraBez);
}

//-----insertBuchung-----
// Bucht die ausgewählten Plätze
// hierfür wird eine Buchung in der Datenbank angelegt und für jeden Platz
// ein Ticket in der Datenbank erstellt

public int insertBuchung(String pVeraID,
                        String pBlock,
                        String pReihe,
                        String pTicketAnzahl)
                        throws java.rmi.RemoteException {

    int vBuchID = 0;
    try{
        // Statements erstellen
        Statement v_statement = vConnection.createStatement();
        Statement v_statement2 = vConnection.createStatement();

        // Sequence für die einzufügenden Buchung einlesen
        ResultSet v_resultset = v_statement.executeQuery
            ("select ID_SEQ.nextval from dual");
        v_resultset.next();
        // Sequence speichern
        vBuchID = v_resultset.getInt(1);

        // Buchungsdatensatz in der DB anlegen
        int vInsertResult = v_statement.executeUpdate("Insert into buchungen"+
            "(buch_id,Buch_Datum) values('"+vBuchID +"',sysdate)");

        // freie Plaetze auslesen
        String vSelect =
            "select plae_id from plaetze, preis_kategorien,preise"+
            " where plae_id not in ( select tick_plae_id from tickets)"+
            " and plae_prka_id = prka_id and prka_id = prei_prka_id "+
            " and prei_vera_id = '"+pVeraID+
            " and plae_block  = '" +pBlock +"'"+
            " and plae_reihe  = " +pReihe;

        ResultSet vResultSetPlaetze = v_statement.executeQuery(vSelect);
    }
}

```

```

// Anzahl der gebuchten Karten
int vAnzahlKarten = 0;
// Anzahl der vom Kunden gewünschten Karten
int vAnzahlKartenSoll = (int) (new Integer(
                                (String) pTicketAnzahl)).intValue();

// Wenn noch mehr Karten gebucht werden müssen
while (vAnzahlKarten < vAnzahlKartenSoll)
{
    // Nächsten Platz ermitteln
    vResultSetPlaetze.next();
    int vPlatzID = vResultSetPlaetze.getInt(1);

    // Sequence für die einzufügenden Tickets einlesen
    ResultSet vResultSetSEQ = v_statement2.executeQuery(
                                "select ID_SEQ.nextval from dual");
    vResultSetSEQ.next();
    int vTickID = vResultSetSEQ.getInt(1);

    // für jeden Platz der Buchung ein Ticket anlegen
    String vInsert="Insert into tickets(tick_id,tick_nr,tick_buch_id,"+
                    "tick_plae_id,tick_vera_id)"+" values("+vTickID +
                    ",to_char("+vTickID+"), "+vBuchID+", "+vPlatzID+", "
                    +pVeraID +")";

    int vInsertResultInsert = v_statement2.executeUpdate(vInsert);

    vAnzahlKarten = vAnzahlKarten +1;
}

// Statements schließen
v_statement.close();
v_statement2.close();

} catch (Exception e){
    System.out.println("Fehler bei insertBuchung:"+e);
}

// Buchungs ID zurückgeben
return vBuchID;
}

//-----insertNeuenKunde-----
// legt einen neuen Kundendatensatz in der Datenbank an

public String insertNeuenKunde(String pKund_Nachame,
                               String pKund_Strasse,
                               String pKund_PLZ,
                               String pKund_Ort,
                               String pKund_Email,
                               String pKund_Kreditkarte_Typ,
                               String pKund_Kreditkarte_Nr,
                               String pKund_Kreditkarte_Guelteig_Bis,
                               String pKund_PWD )
    throws java.rmi.RemoteException{

    int vKund_ID = 0;
    try{
        // Statement erstellen
        Statement v_statement = vConnection.createStatement();

```

```

// Sequence für die einzufügenden Buchung holen
ResultSet v_resultset = v_statement.executeQuery(
    "select ID_SEQ.nextval from dual");
v_resultset.next();
vKund_ID = v_resultset.getInt(1);

// Kundendatensatz anlegen
int vInsertResult = v_statement.executeUpdate(
    "Insert into Kunden( "+
    "kund_id,kund_nr,Kund_Name,Kund_Strasse,"+
    "Kund_PLZ,Kund_Ort,Kund_Email,"+
    "Kund_Kreditkarte_Typ,Kund_Kreditkarte_Nr,"+
    "Kund_Kreditkarte_Guelteig_Bis,Kund_PWD)" +
    " values (" +vKund_ID+", "+vKund_ID+", '"+
    pKund_Nachame+"', '"+pKund_Strasse+"', '"+
    pKund_PLZ+"', '"+pKund_Ort+"', '"+pKund_Email+
    "' , '"+pKund_Kreditkarte_Typ+"', '"+
    pKund_Kreditkarte_Nr+"', '"+
    pKund_Kreditkarte_Guelteig_Bis+"', '"+
    pKund_PWD +"' )");

// Statement schließen
v_statement.close();

} catch (Exception e){
    System.out.println("Fehler bei insertNeuenKunde:"+e);
}

// Rückgabe der KundenID
return (new Integer (vKund_ID)).toString() ;
}

//-----updateKunde-----
// modifiziert einen Kundendatensatz

public String updateKunde(String pKund_Nr,
    String pkund_name,
    String pKund_Strasse,
    String pKund_PLZ,
    String pKund_Ort,
    String pKund_Email,
    String pKund_Kreditkarte_Typ,
    String pKund_Kreditkarte_Nr,
    String pKund_Kreditkarte_Guelteig_Bis,
    String pKund_PWD )
    throws java.rmi.RemoteException{

int vUpdateResult = 0;
try{
    // Statement erstellen
    Statement v_statement = vConnection.createStatement();

    // Kundendatensatz ändern
    String vUpdate = "Update Kunden set kund_name = '"+pkund_name+
        "',Kund_Strasse='"+ pKund_Strasse +
        "',Kund_PLZ='"+pKund_PLZ+
        "',Kund_Ort='"+pKund_Ort+
        "',Kund_Email='"+pKund_Email+
        "',Kund_Kreditkarte_Typ='"+pKund_Kreditkarte_Typ+
        "',Kund_Kreditkarte_Nr='"+pKund_Kreditkarte_Nr+

```

```

        "','Kund_Kreditkarte_Gueltig_Bis='"+
        pKund_Kreditkarte_Gueltig_Bis+
        "','Kund_PWD='"+pKund_PWD+"' where kund_id='"+pKund_Nr;
vUpdateResult = v_statement.executeUpdate(vUpdate);

// Statement schließen
v_statement.close();

} catch (Exception e){
    System.out.println("Fehler bei insertNeuenKunde:"+e);
}

return (new Integer (vUpdateResult)).toString() ;
}

//-----ordneBuchungZuKunde-----
// Ordnet eine Buchung einem bestimmten Kunden zu
public int ordneBuchungZuKunde(String pKund_ID, String pBuch_ID)
    throws java.rmi.RemoteException{

    int vUpdateResult = 0;
    try{
        // Statement erstellen
        Statement v_statement = vConnection.createStatement();

        // Kundenverweis in Buchung eintragen
        vUpdateResult = v_statement.executeUpdate("Update Buchungen "+
            "SET BUCH_KUND_ID='"+ pKund_ID
            +" where BUCH_ID='"+pBuch_ID);

        // Statement schließen
        v_statement.close();

    } catch (Exception e){
        System.out.println("Fehler bei orndeKundeZuBuchung:"+e);
    }
    // Ergebnis der Änderung zurückgeben
    return (vUpdateResult) ;
}

//-----getKundendaten-----
// Liest anhand der Kunden ID und dem Kunden PWD die Kundendaten aus der
// Datenbank und gibt sie als Vektor zurück

public Vector getKundendaten(String pKundID, String pKundPWD)
    throws java.rmi.RemoteException{

    // Rückgabevektor
    Vector vSpalten = new Vector();

    try{
        // Statement erstellen
        Statement v_statement = vConnection.createStatement();

        // Select ausführen
        ResultSet v_resultset = v_statement.executeQuery("SELECT kund_name, "+
            "KUND_ORT, KUND_EMAIL, KUND_PLZ, KUND_STRASSE, KUND_KREDITKARTE_TYP, "+
            "KUND_KREDITKARTE_NR, KUND_KREDITKARTE_GUELTIK_BIS, KUND_PWD "+
            " FROM KUNDEN where KUND_ID='"+pKundID+" AND KUND_PWD='"+
            pKundPWD+"'");
    }

```



```

// Ergebnismenge auswerten
while ( v_resultset.next() ) {
    vSpalten.addElement(v_resultset.getString(1));
    vSpalten.addElement(v_resultset.getString(2));
    vSpalten.addElement(v_resultset.getString(3));
    vSpalten.addElement(v_resultset.getString(4));
    vSpalten.addElement(v_resultset.getString(5));
    vSpalten.addElement(v_resultset.getString(6));
    vSpalten.addElement(v_resultset.getString(7));
    vSpalten.addElement(v_resultset.getString(8));
    vSpalten.addElement(v_resultset.getString(9));
}

// Statement schließen
v_statement.close();

} catch (Exception e){
    System.out.println("Fehler bei getKundendaten:"+e);
}

// Vektor zurückgeben
return vSpalten;
}

//-----deleteBuchung-----
// Löschen einer Buchung
// Zusätzlich werden alle mit der Buchung verbundnen Tickets gelöscht
// bzw. die Plätze wieder freigeben

public int deleteBuchung(String pBuchID) throws java.rmi.RemoteException{

    int vResult = 0;

    try{
        // Statement erstellen
        Statement v_statement = vConnection.createStatement();

        // Delete ausführen
        // alle Tickets der Buchung löschen
        int vDeleteResult = v_statement.executeUpdate(
            "DELETE TICKETS WHERE Tick_BUCH_ID =" + pBuchID);

        // Buchung löschen
        int vDeleteResult2 = v_statement.executeUpdate(
            "DELETE Buchungen WHERE BUCH_ID =" + pBuchID);
        vResult = vDeleteResult2 + vDeleteResult;
        // Statement schließen
        v_statement.close();

    } catch (Exception e){
        System.out.println("Fehler bei deleteBuchung:"+e);
    }

    // Ergebnis der Löschung zurückgeben
    return vResult;
}

//-----getBuchungInfo-----
// liefert für die übergebende Buchung Informationen in Form eines
// Strings zurück
// Dabei kann die Art der Information durch den Parameter pModus bestimmt

```

```

// werden
// pModus = "Screen" -> aufbereitet für die Anzeige in der Maske
//      = "Email" -> aufbereitet als E-Mail Text (Buchungsbestätigung)
public String getBuchungInfo(String pBuchID,String pVeraID, String pModus)
    throws java.rmi.RemoteException{

    String vBuchungsInfo = "";

    // Format für Zahlenwerte definieren
    NumberFormat vNumberFormat =
        NumberFormat.getCurrencyInstance(Locale.GERMANY);

    String vKundId = "";

    try{
        // Statement erstellen
        Statement v_statement = vConnection.createStatement();

        // Select ausführen
        ResultSet v_resultset = v_statement.executeQuery(
            "select to_char(PLAE_REIHE,'000'),"+
            "to_char(to_number(PLAE_BLOCK),'000')"+
            ",to_char(PLAE_NR,'000'),PRKA_BEZEICHNUNG,PREI_PREIS, "+
            "ltrim(to_char(rownum,'00')), PREI_PREIS,buch_Kund_id "+
            " from buchungen,tickets,plaetze,Preis_kategorien,preise"+
            " where buch_ID      = tick_buch_id"+
            " and   tick_plae_id =   plae_id"+
            " and   prka_id      = plae_prka_id"+
            " and   prei_prka_id = prka_id"+
            " and   prei_vera_id = tick_vera_id"+
            " and   buch_ID      = "+pBuchID+" order by PLAE_NR ");

        String vVeraInfo = getVeranstaltungInfo(pVeraID, "Daten");
        String vVeraName = getVeranstaltungInfo(pVeraID, "Name");

        // Informationen erstellen
        if (pModus.equals("Screen"))
            vBuchungsInfo = "Buchungsinformationen:\n\n"+
                "Veranstaltung:\n"+vVeraName+ "\n"+vVeraInfo+"\n\nPlätze:\n";
        if (pModus.equals("Email"))
            vBuchungsInfo =
                "Sehr geehrter Kunde,\n\nwir bedanken uns für Ihre "+
                "Buchung und übersenden Ihnen hiermit \nfolgende "+
                "Buchungsbestätigung:\n\n"+
                "Veranstaltung:\n"+vVeraName+ "\n"+vVeraInfo+
                "\n\nBuchungsnummer: "+pBuchID;

        // Ergebnismenge in lokale Variablen schreiben
        double vGesamtpreis = 4.97;// 4.97 = Bearbeitungsgebühren
        String vAusgleich = "";
        int vZaehler = 1;

        // Ergebnismenge auswerten
        while (v_resultset.next() ) {

            if ((vZaehler == 1)&&(pModus.equals("Email"))){
                vBuchungsInfo = vBuchungsInfo + "\nKunden-Nr: "+
                    v_resultset.getString(8)+"\n\nPlätze:\n";
                vZaehler = 2;
            }
            // wenn der Zahlen wert > 99.99 dann muß ein Ausgleichszeichen
            // eingefügt werden

```

```

        if (v_resultset.getDouble(5) > 99.99)
            vAusgleich = "";
        else
            vAusgleich = " ";

        vGesamtpreis = vGesamtpreis + v_resultset.getDouble(7);
        vBuchungsInfo = vBuchungsInfo + v_resultset.getString(6) + ". Block:" +
            v_resultset.getString(2) + " Reihe:" +
            v_resultset.getString(1) + " Platz:" +
            v_resultset.getString(3) + " Preis: " +
            vAusgleich + vNumberFormat.format(v_resultset.getFloat(5))
            + " (" + v_resultset.getString(4) + ") \n";
    }

    // Informationen erstellen
    vBuchungsInfo = vBuchungsInfo + "\nzzgl. Bearbeitungsgebühren      " +
        "                                4,97 DM" ;

    vBuchungsInfo = vBuchungsInfo + "\n-----" +
        "-----" ;

    vBuchungsInfo = vBuchungsInfo + "\nGesamtpreis                " +
        "                " + vNumberFormat.format(vGesamtpreis) ;

    if (pModus.equals("Email"))
        vBuchungsInfo = vBuchungsInfo + "\n\nDer Gesamtpreis von " +
            vNumberFormat.format(vGesamtpreis) +
            " wird von Ihrer Kreditkarte abgebucht.\n" +
            "Wir wünschen Ihnen eine erlebnisreiche Veranstaltung." +
            " \n\nIhr OnlineTicket-Team\n\nOnlineTicket " +
            "GmbH\nKölner Str. 78a \n50678 Köln\nTel.: 0221/600151\n";

    //Statement schließen
    v_statement.close();

} catch (Exception e){
    System.out.println("Fehler bei getBuchungInfo:"+e);
}

// Information zurückgeben
return (vBuchungsInfo);
}

//-----sendEmail-----
// Versendet eine Email mit dem Programm "qmail"
public void sendEmail(String vAdressAutor,
                     String vAdressEmpfaenger,
                     String vBetreff, String vText)
    throws java.rmi.RemoteException {

    String vTextdatei = "email.dat";
    try {
        // Datei mit Email-Text erstellen
        FileWriter vFileEmail = new FileWriter (vTextdatei);
        vFileEmail.write(vText);
        vFileEmail.close();

        // Holen des Runtime Objektes
        Runtime runtime = Runtime.getRuntime();

        // Holen eines Prozeß Thread des Operating Systems und
        // Ausführen von qmail
        String vAufruf = "c:/aktuell/Email/qmail.exe " +
            vAdressEmpfaenger+

```

```

        " /h=mail.opitz-partner.de "+
        " /f="+vAdressAutor +
        " \"\" +vBetreff + "\""+
        " \"\"+vTextdatei+ "\"";

        Process process = runtime.exec(vAufruf );
    }

    catch (Exception ioe) {
        System.out.println("Error sending Email: " + ioe);
    }
}

//----- getVeranstaltungsortBild-----

public byte[] getVeranstaltungsortBild(String pVeorId)
    throws java.rmi.RemoteException{
    ResultSet vRset = null;
    BLOB vBlobOut = null;

    try{
        // Abfrage erstellen
        Statement vStatement = vConnection.createStatement();

        // Auslesen des Bildes aus der DB
        vRset = vStatement.executeQuery("SELECT veor_bild FROM "+
            "veranstaltung_orte, veranstaltungen"+
            " WHERE vera_veor_id = veor_id "+
            "and vera_ID = " + pVeorId);

        while (vRset.next()){
            vBlobOut = ((OracleResultSet)vRset).getBLOB(1);
        }

        // Bild in ein Byte-Array umwandeln
        byte [] byte_array = new byte [(int)vBlobOut.length()];
        byte_array = vBlobOut.getBytes (1, (int)vBlobOut.length());

        return byte_array;
    }
    catch (Exception e){
        System.out.println("Error in getVeranstaltungBild: " + e);
        return null;
    }
}

}
//-----Ende der Bean-Klasse Ticket-----

```

HTML-Dokument zum Laden des Applet

```
<EMBED type = "application/x-java-applet;version=1.2"
  java_CODE = "client.AppletTicketShop.class"
  java_CODEBASE = "."
  java_ARCHIVE = "ejbapi.jar,_client.jar,oasoorb.jar,javax-ssl-1_2.jar,
                jssl-1_2.jar,TicketClient.jar "
  WIDTH=800
  HEIGHT=600
  org.omg.CORBA.ORBClass = "oracle.oas.orb.CORBA.ORB"
  org.omg.CORBA.ORBSingletonClass = "oracle.oas.orb.CORBA.ORB"
  ORBdisableLocator = "true"
  pluginspage = "http://java.sun.com/products/plugin/1.2/plugin-
install.html">
</EMBED>
```

Quellcode der PL/SQL-Prozedur "delete_Reservierungen"

```
procedure delete_Reservierungen IS
BEGIN
  -- Selektieren der Reservierungen, die älter sind als 15 Minuten
  for rec in ( select buch_id
                from buchungen
                where to_char(sysdate,'YYYYMMDDHH24MI') -
                    to_char(buch_datum,'YYYYMMDDHH24MI') > 15 and buch_kund_id is null)
  loop
    -- alle Tickets der Reservierung löschen
    delete tickets where tick_buch_id = rec.buch_id;
    -- Buchung löschen
    delete buchungen where buch_id = rec.buch_id;
  end loop;
END;
```

Skript zum Erzeugen des Datenbank-Schema

```
PROMPT Creating Table 'PREIS_KATEGORIEN'
CREATE TABLE PREIS_KATEGORIEN
  (PRKA_ID NUMBER(38) NOT NULL
  ,PRKA_BEZEICHNUNG VARCHAR2(50) NOT NULL
  ,PRKA_VEOR_ID NUMBER(38) NOT NULL
  )
/

PROMPT Creating Table 'PLAETZE'
CREATE TABLE PLAETZE
  (PLAE_ID NUMBER(38) NOT NULL
  ,PLAE_NR NUMBER(38) NOT NULL
  ,PLAE_REIHE NUMBER(5)
  ,PLAE_BLOCK VARCHAR2(5)
  ,PLAE_PRKA_ID NUMBER(38) NOT NULL
  )
```

```
/

PROMPT Creating Table 'VERANSTALTUNG_ORTE'
CREATE TABLE VERANSTALTUNG_ORTE
  (VEOR_STRASSE VARCHAR2(100)
  ,VEOR_ID NUMBER(38) NOT NULL
  ,VEOR_BEZEICHNUNG VARCHAR2(30) NOT NULL
  ,VEOR_ORT CHAR(50)
  ,VEOR_PLZ NUMBER(5,0)
  ,VEOR_BILD BLOB
  )
/

PROMPT Creating Table 'KUNDEN'
CREATE TABLE KUNDEN
  (KUND_ID NUMBER(38) NOT NULL
  ,KUND_NR NUMBER(38) NOT NULL
  ,KUND_NAME VARCHAR2(50) NOT NULL
  ,KUND_ORT VARCHAR2(50)
  ,KUND_PLZ VARCHAR2(5)
  ,KUND_STRASSE VARCHAR2(50)
  ,KUND_KREDITKARTE_TYP VARCHAR2(10)
  ,KUND_KREDITKARTE_NR VARCHAR2(20)
  ,KUND_KREDITKARTE_GUELTIG_BIS VARCHAR2(6)
  ,KUND_PWD VARCHAR2(15)
  ,KUND_EMAIL VARCHAR2(50)
  )
/

PROMPT Creating Table 'PREISE'
CREATE TABLE PREISE
  (PREI_ID NUMBER(38) NOT NULL
  ,PREI_PRKA_ID NUMBER(38) NOT NULL
  ,PREI_VERA_ID NUMBER(38) NOT NULL
  ,PREI_PREIS NUMBER(5,2)
  )
/

PROMPT Creating Table 'BUCHUNGEN'
CREATE TABLE BUCHUNGEN
  (BUCH_ID NUMBER(38) NOT NULL
  ,BUCH_DATUM DATE
  ,BUCH_KUND_ID NUMBER(38)
  )
/

PROMPT Creating Table 'TICKETS'
CREATE TABLE TICKETS
  (TICK_ID NUMBER(38) NOT NULL
  ,TICK_NR VARCHAR2(20)
  ,TICK_BUCH_ID NUMBER(38)
  ,TICK_PLAE_ID NUMBER(38) NOT NULL
  ,TICK_VERA_ID NUMBER(38) NOT NULL
  )
/

PROMPT Creating Table 'VERANSTALTUNGEN'
CREATE TABLE VERANSTALTUNGEN
  (VERA_ID NUMBER(38) NOT NULL
  ,VERA_INTERPRET VARCHAR2(50)
  ,VERA_BEZEICHNUNG VARCHAR2(50) NOT NULL
  ,VERA_DATUM DATE NOT NULL
```

```
,VERA_BEGINN CHAR(10)
,VERA_BESCHREIBUNG VARCHAR2(2000)
,VERA_VEOR_ID NUMBER(38) NOT NULL
)
/
```

```
PROMPT Creating Primary Key on 'PREIS_KATEGORIEN'
ALTER TABLE PREIS_KATEGORIEN
  ADD CONSTRAINT PRKA_PK PRIMARY KEY
    (PRKA_ID)
/
```

```
PROMPT Creating Primary Key on 'PLAETZE'
ALTER TABLE PLAETZE
  ADD CONSTRAINT PLAT_PK PRIMARY KEY
    (PLAE_ID)
/
```

```
PROMPT Creating Primary Key on 'VERANSTALTUNG_ORTE'
ALTER TABLE VERANSTALTUNG_ORTE
  ADD CONSTRAINT VEOR_PK PRIMARY KEY
    (VEOR_ID)
/
```

```
PROMPT Creating Primary Key on 'KUNDEN'
ALTER TABLE KUNDEN
  ADD CONSTRAINT KUND_PK PRIMARY KEY
    (KUND_ID)
/
```

```
PROMPT Creating Primary Key on 'PREISE'
ALTER TABLE PREISE
  ADD CONSTRAINT PREI_PK PRIMARY KEY
    (PREI_ID)
/
```

```
PROMPT Creating Primary Key on 'BUCHUNGEN'
ALTER TABLE BUCHUNGEN
  ADD CONSTRAINT BUCH_PK PRIMARY KEY
    (BUCH_ID)
/
```

```
PROMPT Creating Primary Key on 'TICKETS'
ALTER TABLE TICKETS
  ADD CONSTRAINT TICK_PK PRIMARY KEY
    (TICK_ID)
/
```

```
PROMPT Creating Primary Key on 'VERANSTALTUNGEN'
ALTER TABLE VERANSTALTUNGEN
  ADD CONSTRAINT VERA_PK PRIMARY KEY
    (VERA_ID)
/
```

```
PROMPT Creating Foreign Keys on 'PREIS_KATEGORIEN'
ALTER TABLE PREIS_KATEGORIEN ADD CONSTRAINT
  PRKA_VEOR_FK FOREIGN KEY
    (PRKA_VEOR_ID) REFERENCES VERANSTALTUNG_ORTE
    (VEOR_ID)
/
```

```
PROMPT Creating Foreign Keys on 'PLAETZE'
ALTER TABLE PLAETZE ADD CONSTRAINT
  PLAT_PRKA_FK FOREIGN KEY
    (PLAE_PRKA_ID) REFERENCES PREIS_KATEGORIEN
    (PRKA_ID)
/
```

```
PROMPT Creating Foreign Keys on 'PREISE'
ALTER TABLE PREISE ADD CONSTRAINT
  PREI_PRKA_FK FOREIGN KEY
    (PREI_PRKA_ID) REFERENCES PREIS_KATEGORIEN
    (PRKA_ID) ADD CONSTRAINT
  PREI_VERA_FK FOREIGN KEY
    (PREI_VERA_ID) REFERENCES VERANSTALTUNGEN
    (VERA_ID)
/
```

```
PROMPT Creating Foreign Keys on 'BUCHUNGEN'
ALTER TABLE BUCHUNGEN ADD CONSTRAINT
  BUCH_KUND_FK FOREIGN KEY
    (BUCH_KUND_ID) REFERENCES KUNDEN
    (KUND_ID)
/
```

```
PROMPT Creating Foreign Keys on 'TICKETS'
ALTER TABLE TICKETS ADD CONSTRAINT
  TICK_PLAT_FK FOREIGN KEY
    (TICK_PLAE_ID) REFERENCES PLAETZE
    (PLAE_ID) ADD CONSTRAINT
  TICK_VERA_FK FOREIGN KEY
    (TICK_VERA_ID) REFERENCES VERANSTALTUNGEN
    (VERA_ID) ADD CONSTRAINT
  TICK_BUCH_FK FOREIGN KEY
    (TICK_BUCH_ID) REFERENCES BUCHUNGEN
    (BUCH_ID)
/
```

```
PROMPT Creating Foreign Keys on 'VERANSTALTUNGEN'
ALTER TABLE VERANSTALTUNGEN ADD CONSTRAINT
  VERA_VEOR_FK FOREIGN KEY
    (VERA_VEOR_ID) REFERENCES VERANSTALTUNG_ORTE
    (VEOR_ID)
/
```

```
CREATE SEQUENCE ID_SEQ START WITH 1 INCREMENT BY 1 MINVALUE 1 NOCACHE
NOCYCLE ;
```


Literaturverzeichnis

Brown, Bradley D.: Oracle8i - Web Development. Osborn/McGraw Hill. Berkeley 2000.

Denninger, Stefan und Ingo Peters: Enterprise JavaBeans. 1. Aufl. München 2000.

Geiger, Erwin: Leistungsfähige Mittler im Netz. In: IT-Journal Ausgabe 06/99.

Herold, Helmut: UNIX-Grundlagen: Kommandos und Konzepte. 3. Aufl. Bonn 1994.

Lück, Wolfgang: Technik des wissenschaftlichen Arbeitens: Seminararbeit, Diplomarbeit, Dissertation. 5. Aufl. München 1997.

Oracle Application Server Documentation, Release 4.0.8.1: Administration Guide.

Oracle Application Server Documentation, Release 4.0.8.1: Developers Guide: EJB, ECO/Java and CORBA Applications.

Oracle Application Server Documentation, Release 4.0.8.1: Developers Guide: C++ CORBA Applications.

Oracle Application Server Documentation, Release 4.0.8.1: Developer's Guide: JServlet Applications.

Oracle Application Server Documentation, Release 4.0.8.1: Developer's Guide: LiveHTML and Perl Applications.

Oracle Application Server Documentation, Release 4.0.8.1: Developer's Guide: PL/SQL and ODBC Applications.

Oracle Application Server Documentation, Release 4.0.8.1: Installation Guide for Windows NT.

Oracle Application Server Documentation, Release 4.0.8.1: Overview and Glossary.

Oracle Application Server Documentation, Release 4.0.8.1: Security Guide.

Sayegh, Andreas: CORBA - Standard, Spezifikation, Entwicklung. 2. Aufl. Köln 1999.

Siple, Matthew: The Complete Guide to Java Database Programming. McGraw Hill. New York 1998.

The Common Object Request Broker: Architecture and Spezifikation, Revision 2.3.1. Oktober 1999.

Wilde, Erik: World Wide Web. Technische Grundlagen. Berlin 1999.